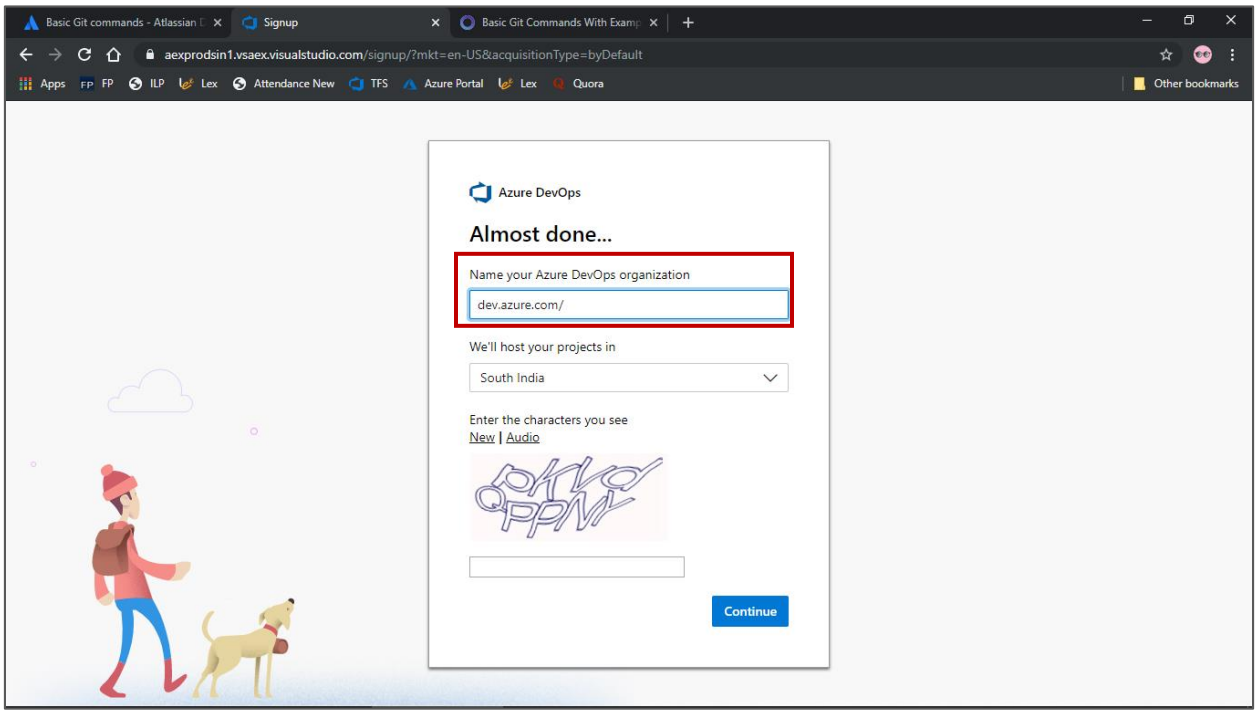
Creating Organization and Project in Azure DevOps

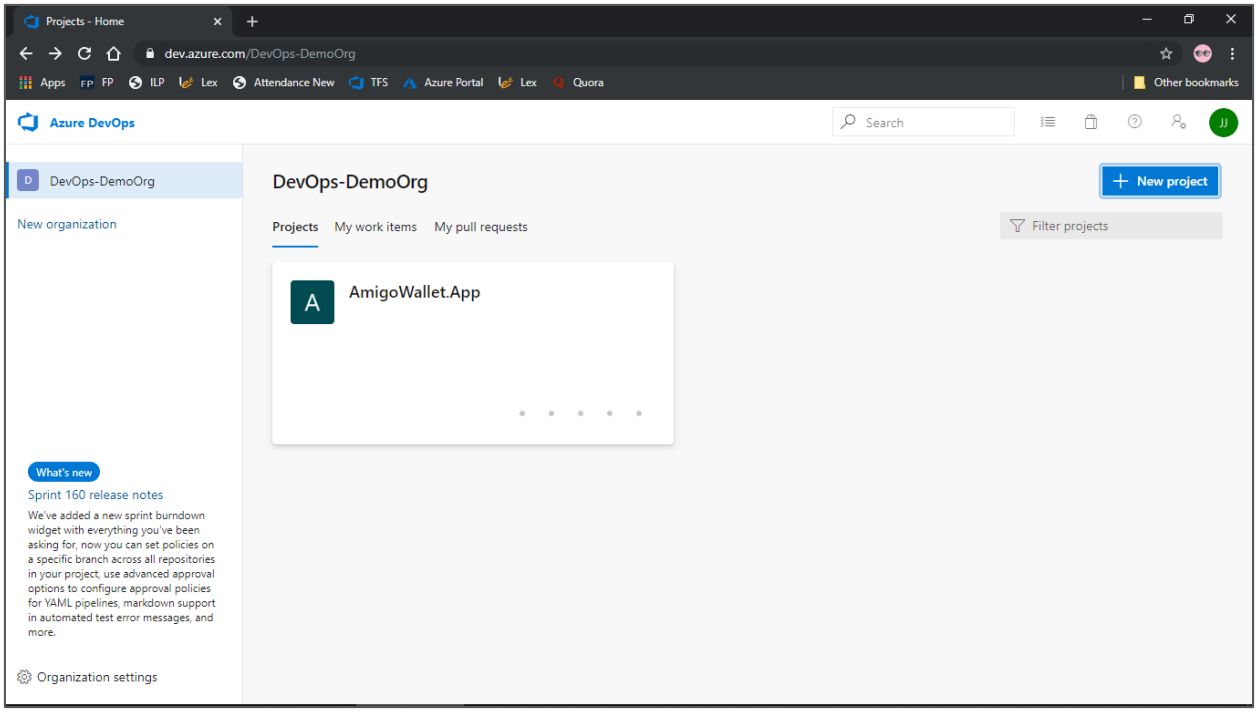
An organization in Azure DevOps is used to connect groups of related projects, helping to scale up an enterprise. With an organization, you gain access to the platform in which you can do the following:

Step 1:

While you are logging in to the DevOps for the 1st time, you'll have to provide an organization name to continue.

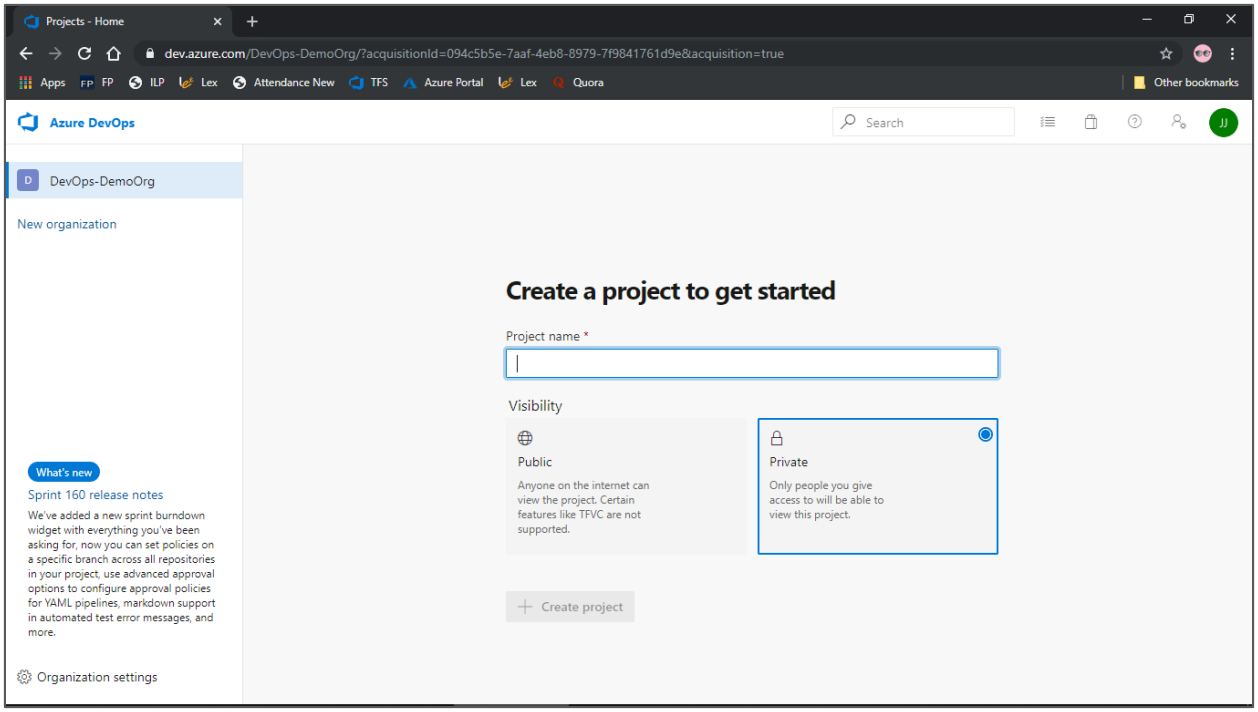


There's also an option to create an organization from the home page of DevOps.

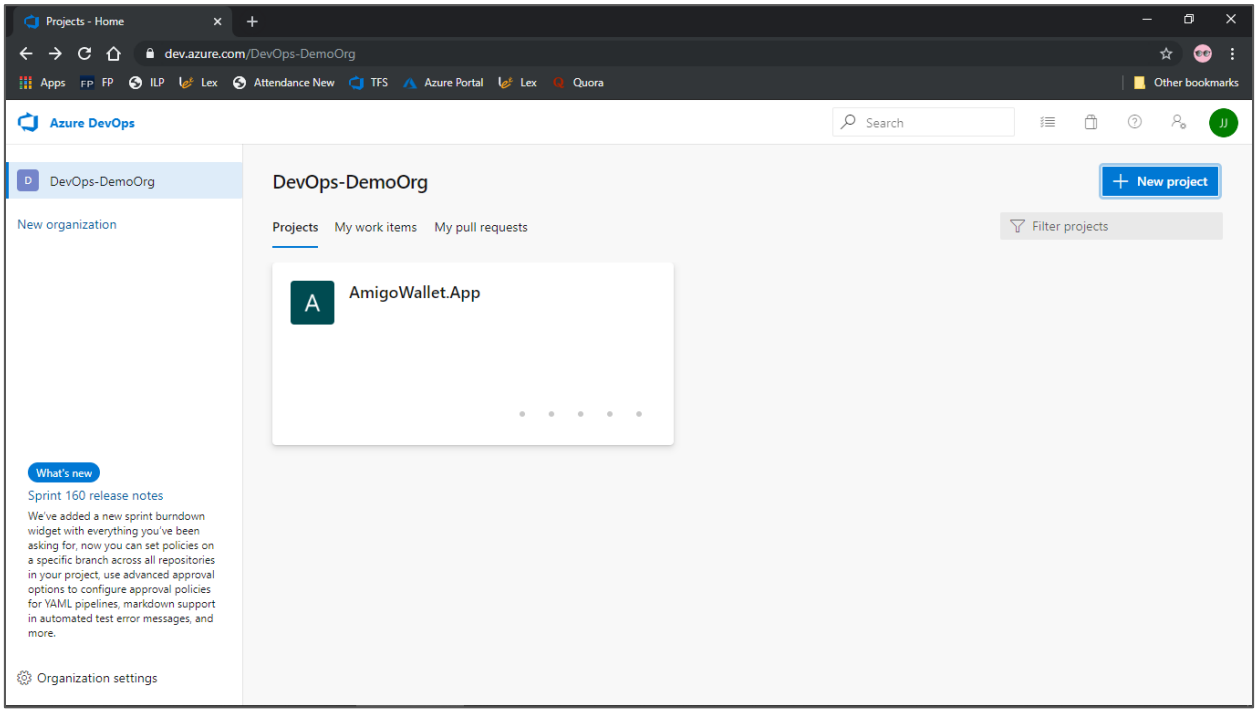


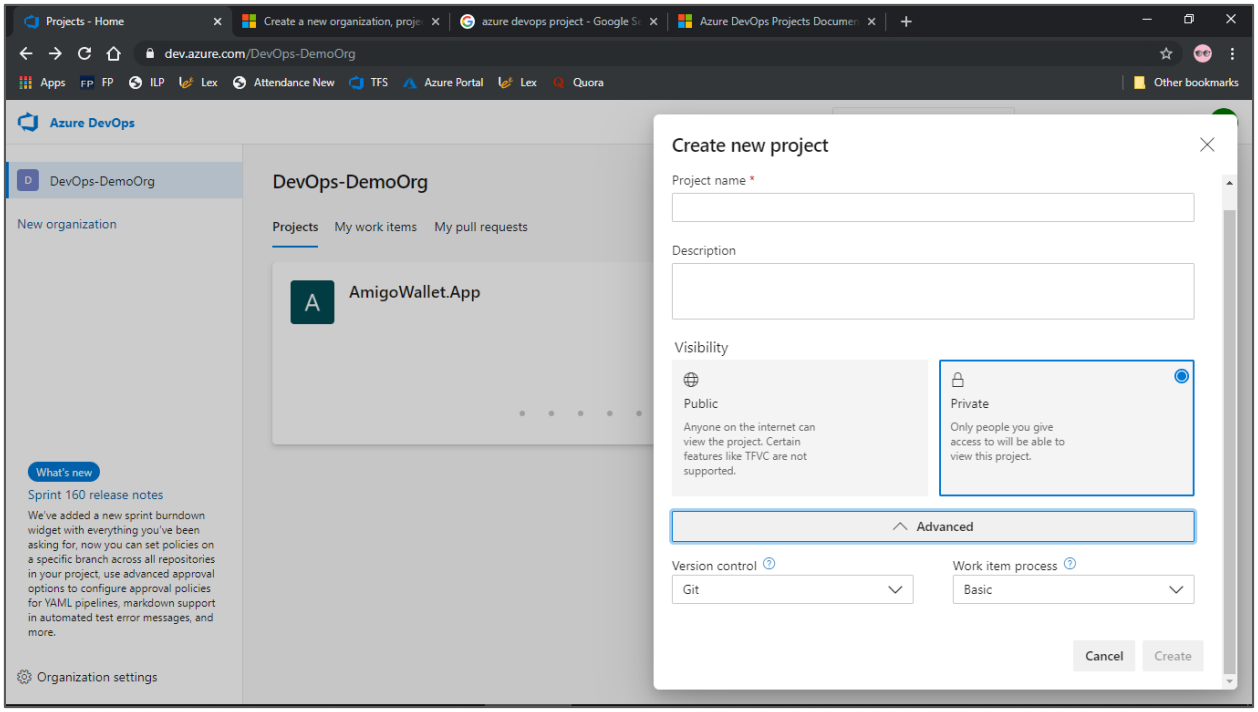
Step 2:

Once you are on the homepage of DevOps, you can view all your organizations on the left pane and all the projects on the center. If you don't have any projects, there'll be an option in the center of the page to create a project. Provide a project name, select the visibility and click on **Create Project** icon to create the project.


Projects - Home
dev.azure.com/DevOps-DemoOrg/?acquisitionId=094c5b5e-7aaf-4eb8-8979-7f9841761d9e&acquisition=true
Azure DevOps
Search
DevOps-DemoOrg
New organization
Create a project to get started
Project name *
Visibility
Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.
Private
Only people you give access to will be able to view this project.
Create project
What's new
Sprint 160 release notes
We've added a new sprint burndown widget with everything you've been asking for, now you can set policies on a specific branch across all repositories in your project, use advanced approval options to configure approval policies for YAML pipelines, markdown support in automated test error messages, and more.
Organization settings

For creating a new project, click on the **New Project** icon on the top right corner. Provide a project name and select the visibility. You can also provide a description, select the desired version control and work item process. Click **Create** to create the project.


Projects - Home
dev.azure.com/DevOps-DemoOrg
Azure DevOps
Search
DevOps-DemoOrg
New organization
DevOps-DemoOrg
New project
Projects
My work items
My pull requests
Filter projects
A
AmigoWallet.App
What's new
Sprint 160 release notes
We've added a new sprint burndown widget with everything you've been asking for, now you can set policies on a specific branch across all repositories in your project, use advanced approval options to configure approval policies for YAML pipelines, markdown support in automated test error messages, and more.
Organization settings



Azure boards

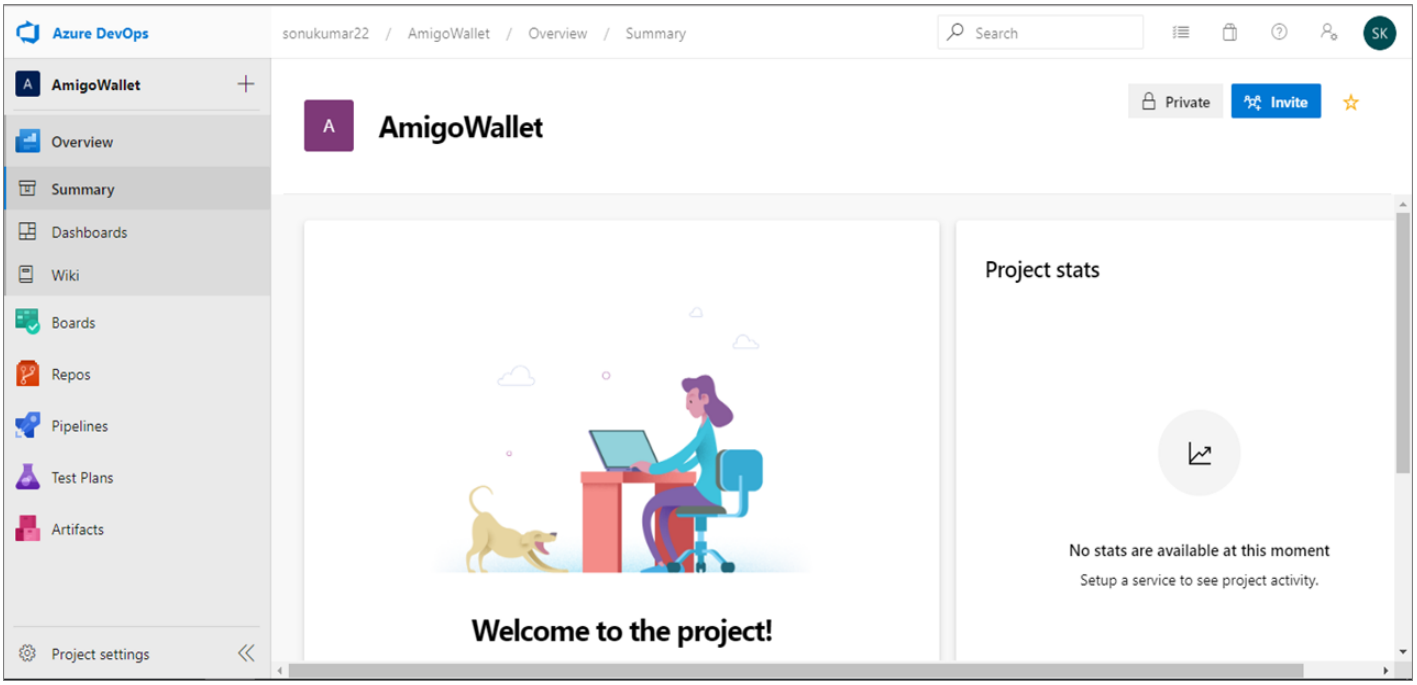
Agile Project Management

Part 1:

Demo Steps: Working with teams, areas and iterations

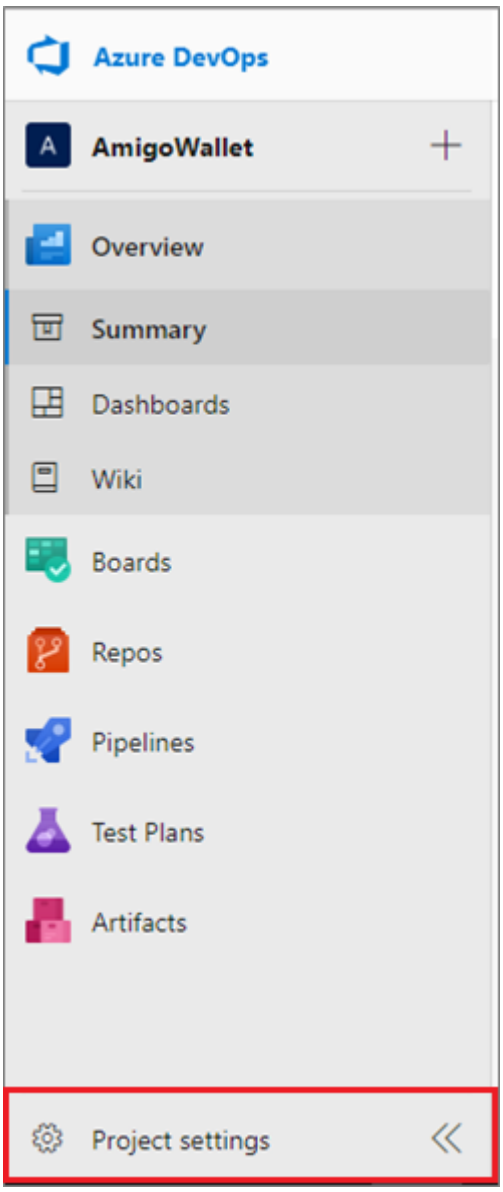
Step 1:

Considering the Project created, you can see the following image:



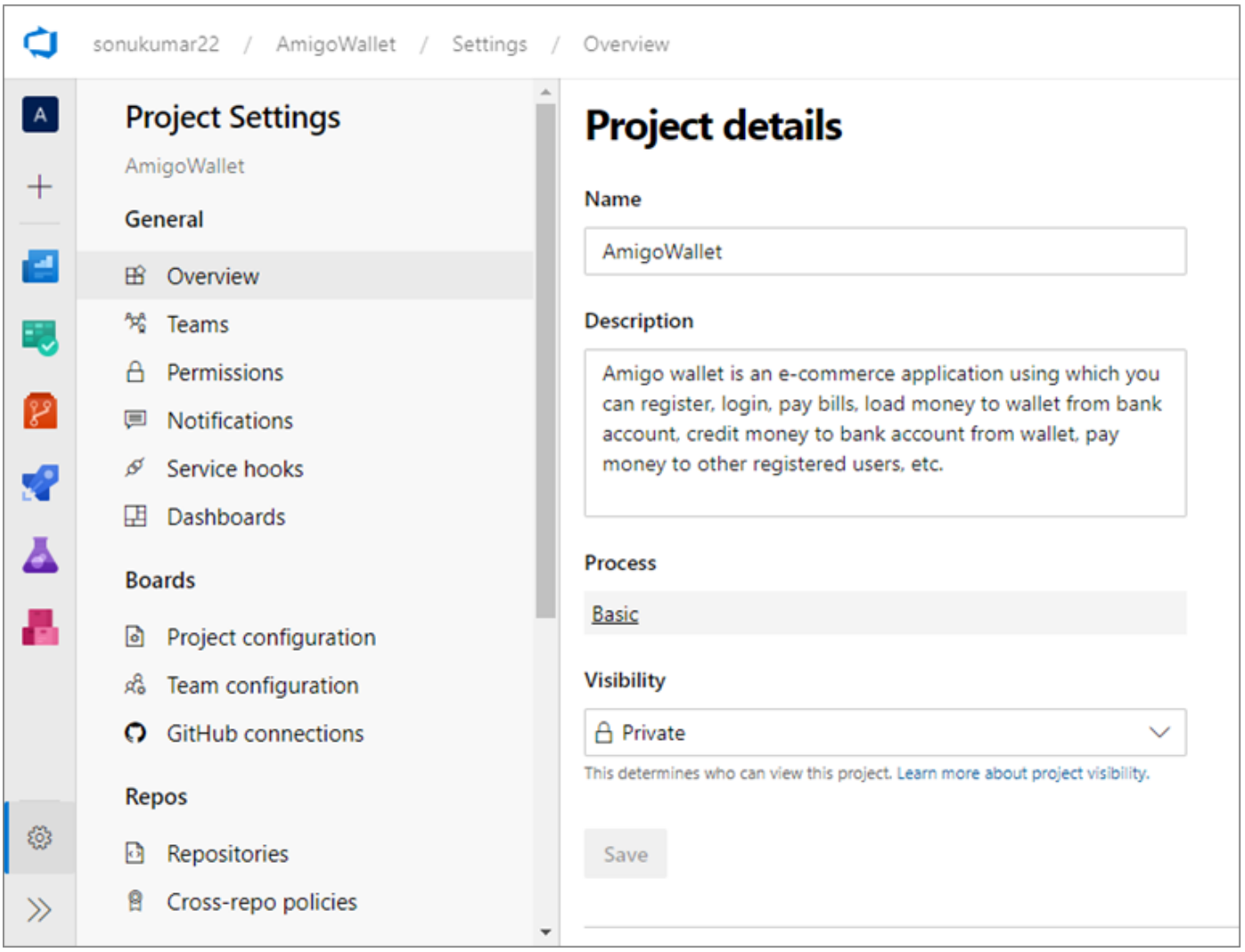
Step 2:

Click on **Project Settings** navigation button to open the settings page:



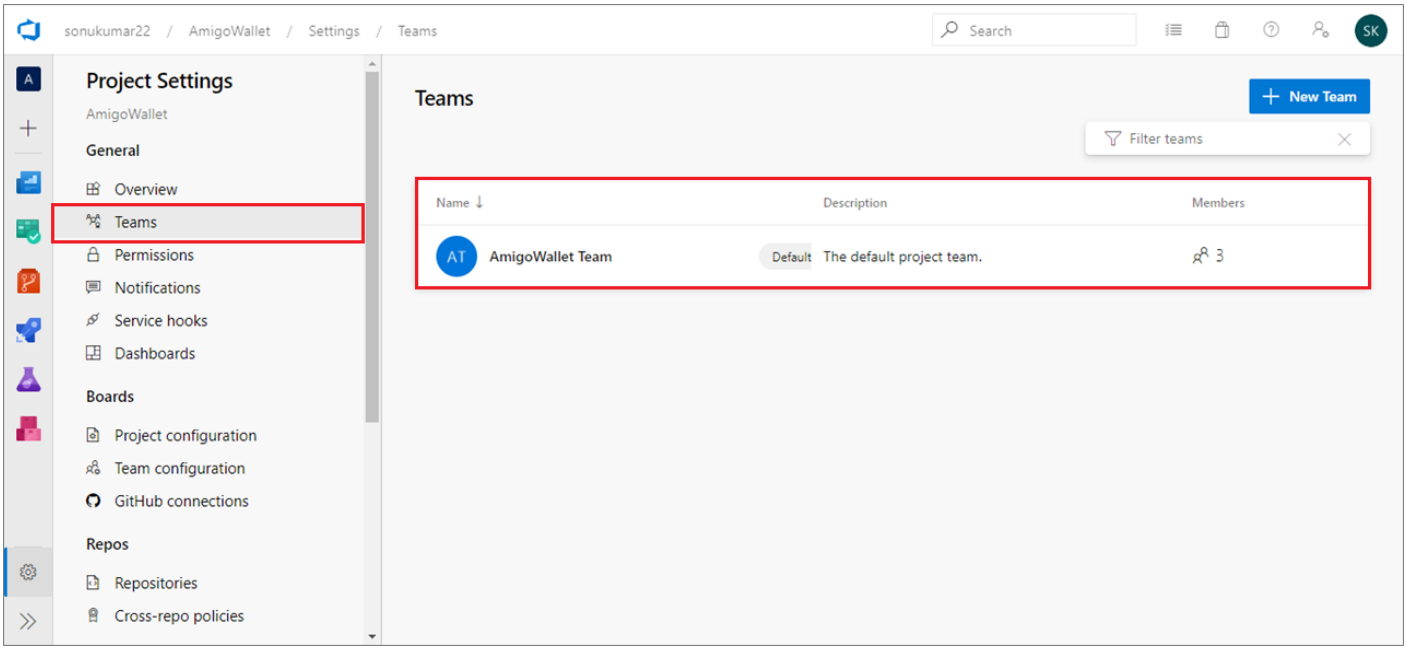
Step 3:

You get **Overview** as the default page which contains the **Project name, Description, Process** and **Visibility**. Fill the required fields with valid input as shown:

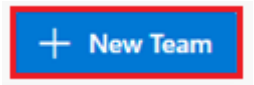


Step 4:

Under Project settings, select **Teams**. You already have the project created. To add new team, click **New Team**:

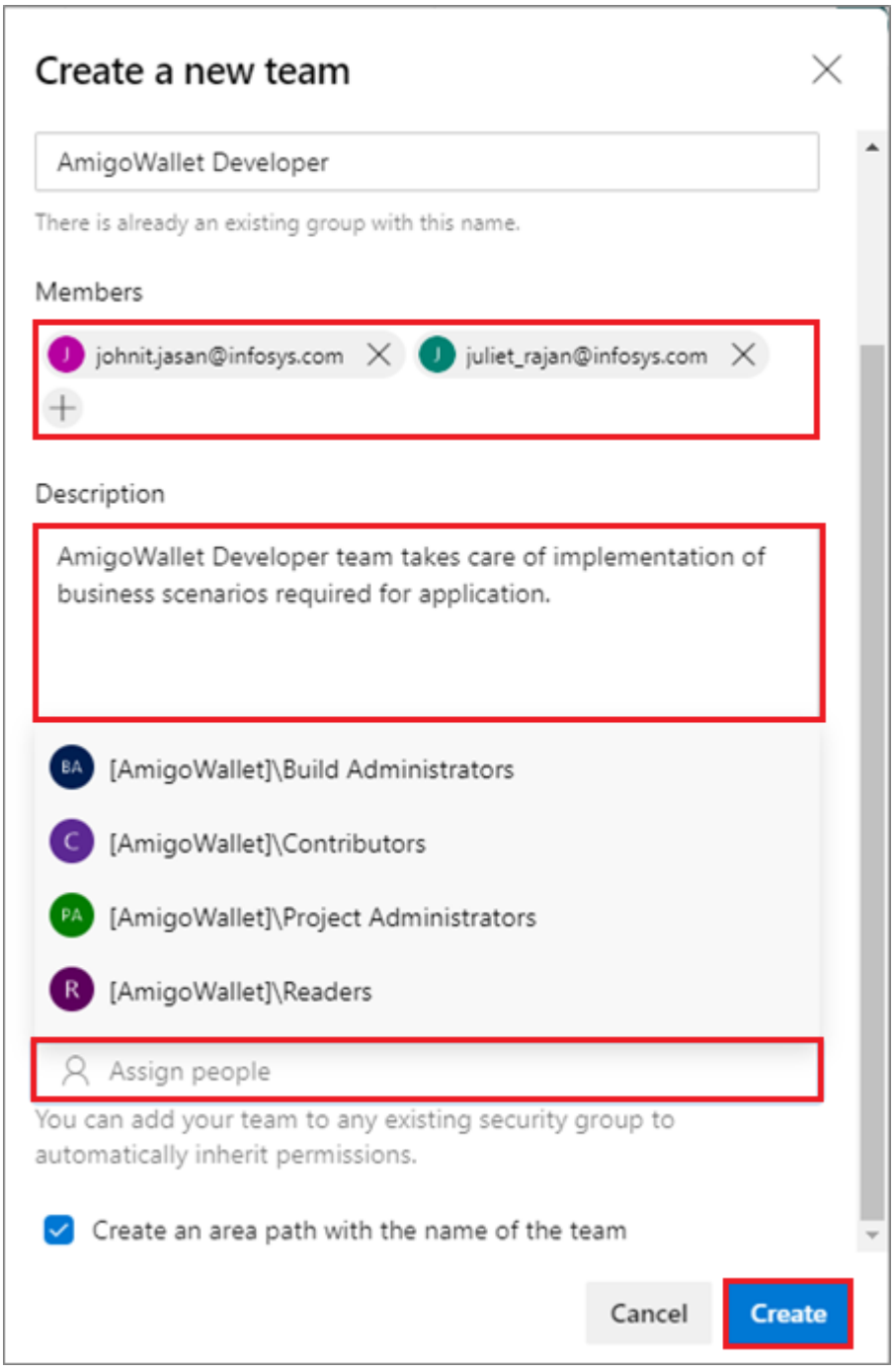


Click on New Team



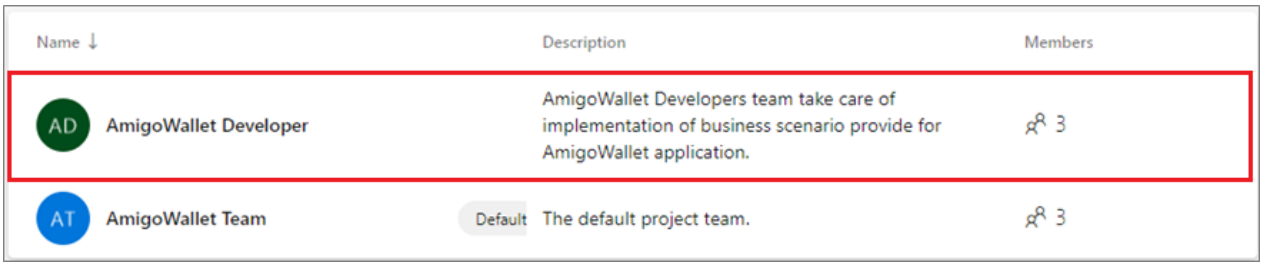
Step 5:

Enter the new name for team, members, description and assign role for member. Click on **Create**



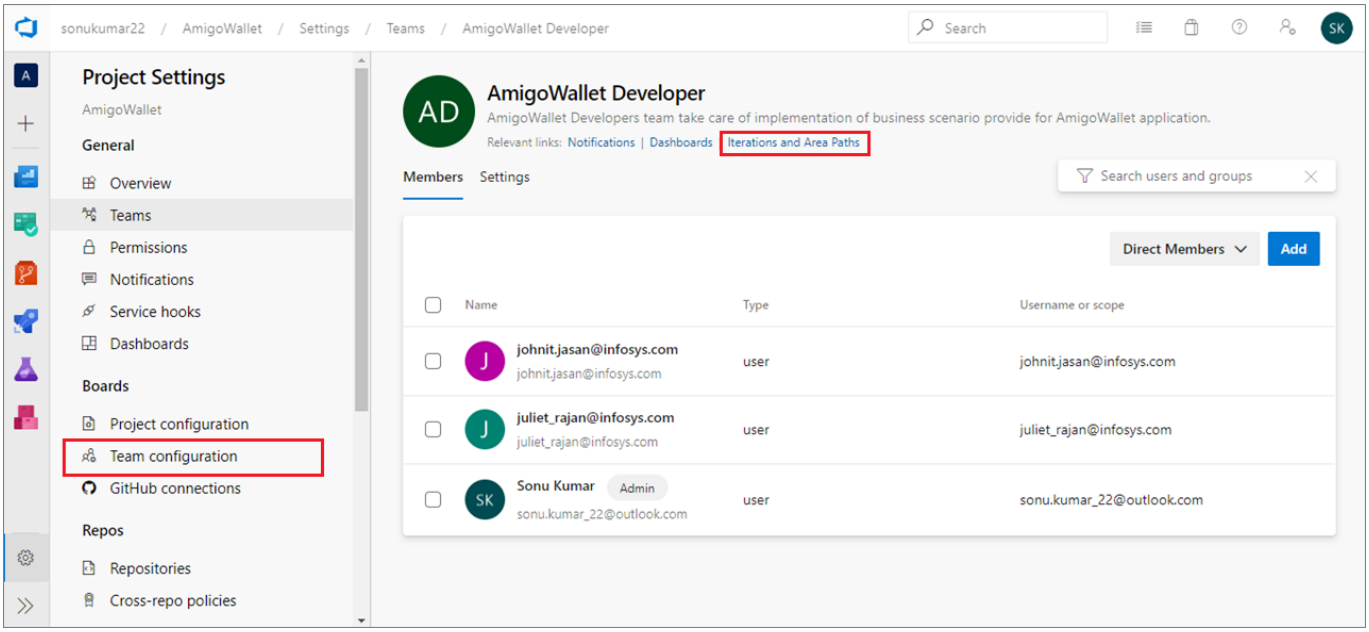
Step 6:

You can see the newly created team AmigoWallet Developer as shown with the fields given as input:



Step 7:

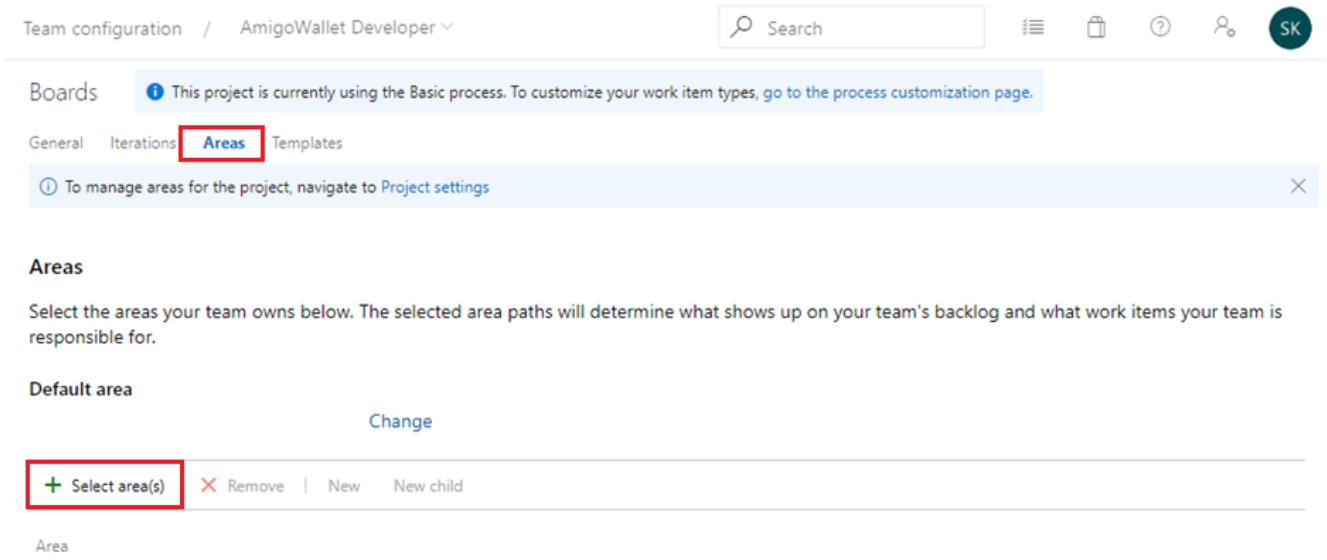
Click on the AmigoWallet Developer team, which navigates to the following page:



You can select **Team Configuration** or **Iterations and Area Paths** to configure iterations and areas.

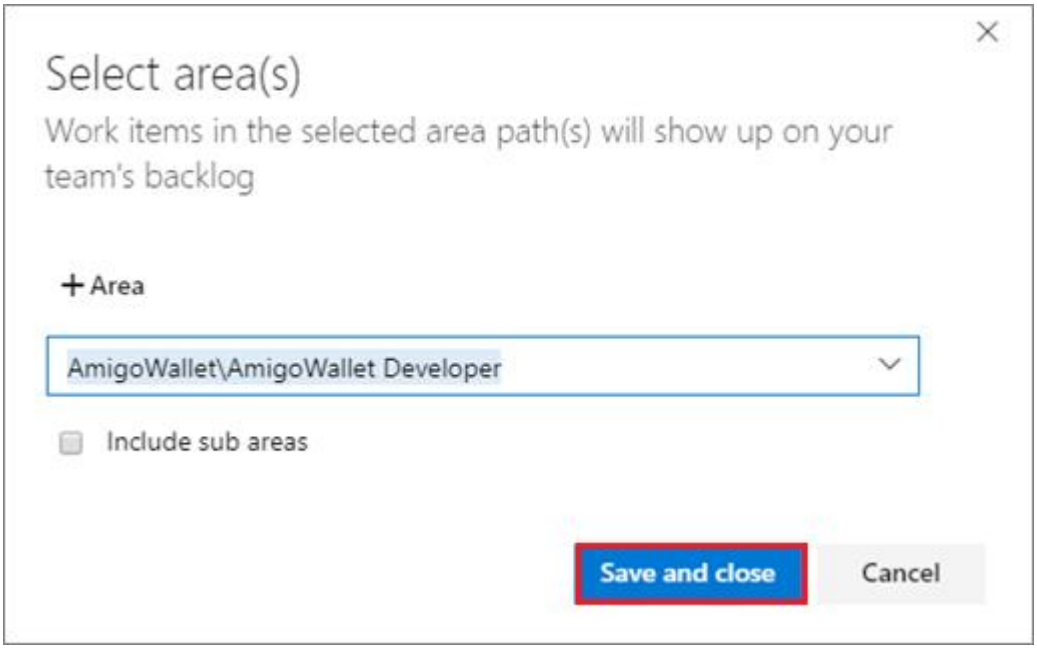
Step 8:

To set the area path, click on **Areas** tab then **Select Area(s)** as shown:


Team configuration / AmigoWallet Developer
Search
SK
Boards
This project is currently using the Basic process. To customize your work item types, go to the process customization page.
General Iterations Areas Templates
To manage areas for the project, navigate to Project settings
Areas
Select the areas your team owns below. The selected area paths will determine what shows up on your team's backlog and what work items your team is responsible for.
Default area
Change
Select area(s)
Remove
New
New child
Area

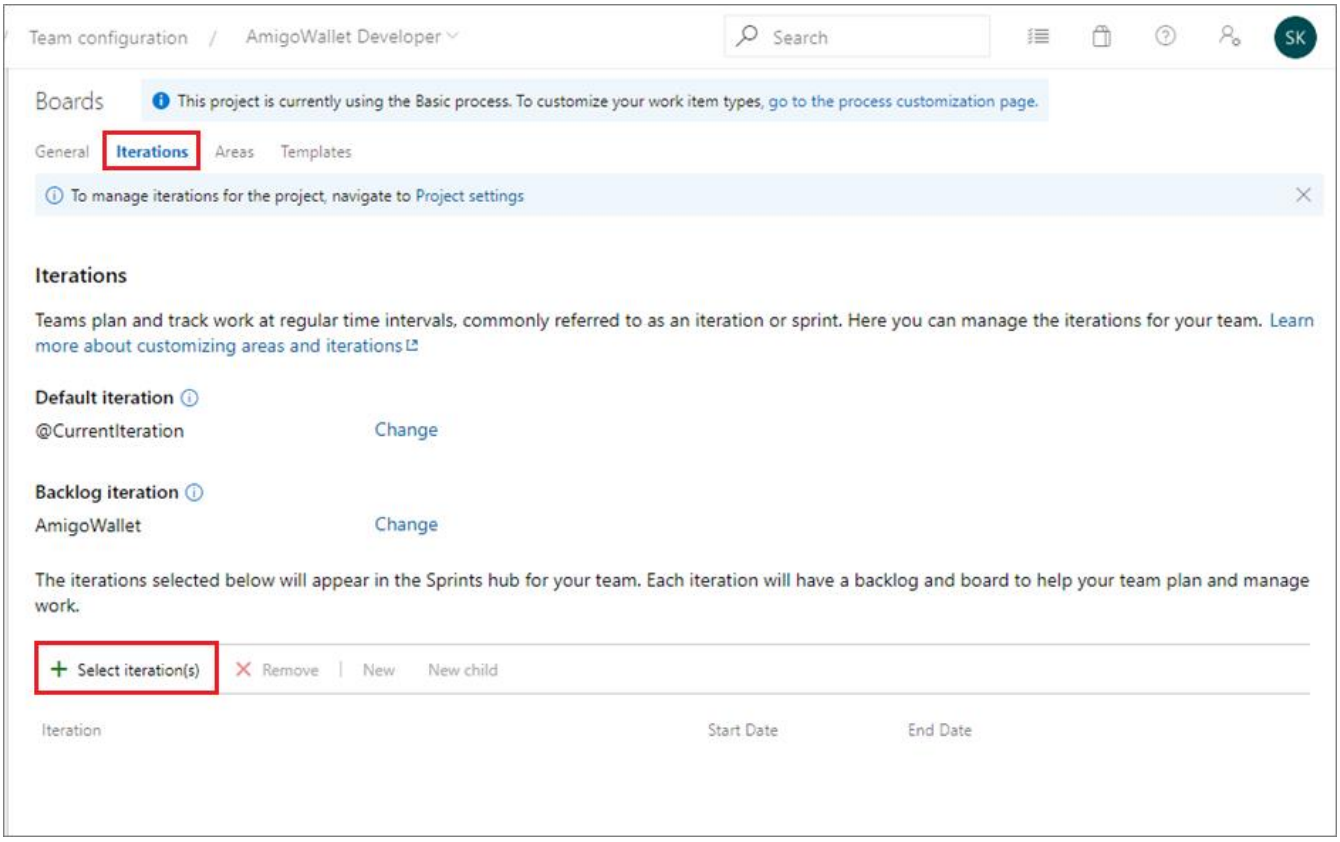
Step 9:

From the drop down available select **Amigo Wallet\AmigoWallet Developer** and click on **Save and Close** button as shown:



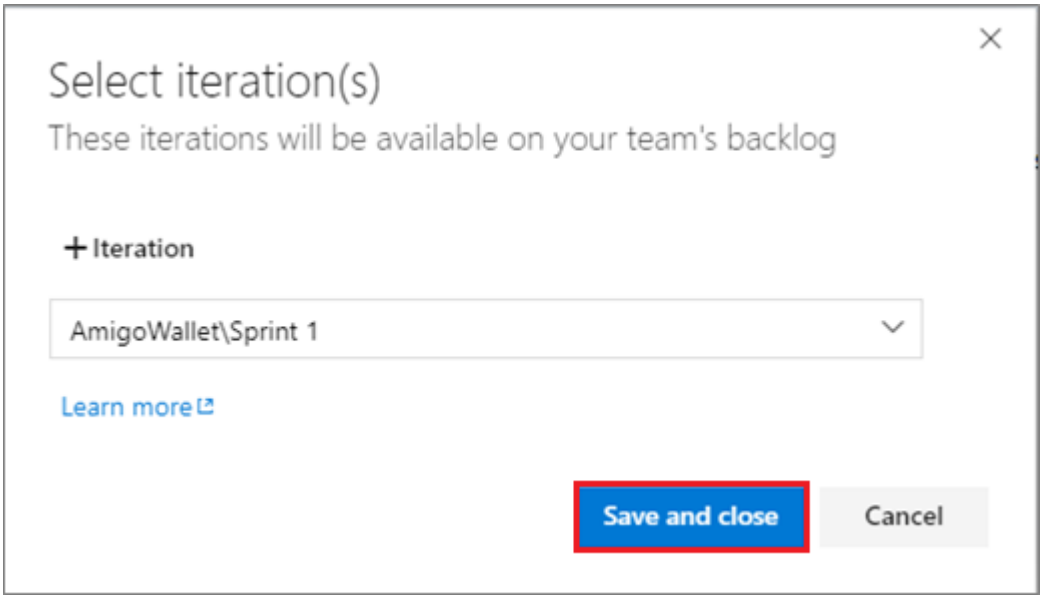
Step 10:

Select **Iterations** tab and click **Select iterations.**


Team configuration / AmigoWallet Developer
Search
SK
Boards
This project is currently using the Basic process. To customize your work item types, go to the process customization page.
General
Iterations
Areas
Templates
To manage iterations for the project, navigate to Project settings
Iterations
Teams plan and track work at regular time intervals, commonly referred to as an iteration or sprint. Here you can manage the iterations for your team. Learn more about customizing areas and iterations
Default iteration
@CurrentIteration
Change
Backlog iteration
AmigoWallet
Change
The iterations selected below will appear in the Sprints hub for your team. Each iteration will have a backlog and board to help your team plan and manage work.
Select iteration(s)
Remove
New
New child
Iteration
Start Date
End Date

Step 11:

Select **AmigoWallet\Sprint 1** and click **Save and Close.** Note that this first sprint has already created. This is because the demo data generator is designed to build out project history so that this sprint occurs in the past.

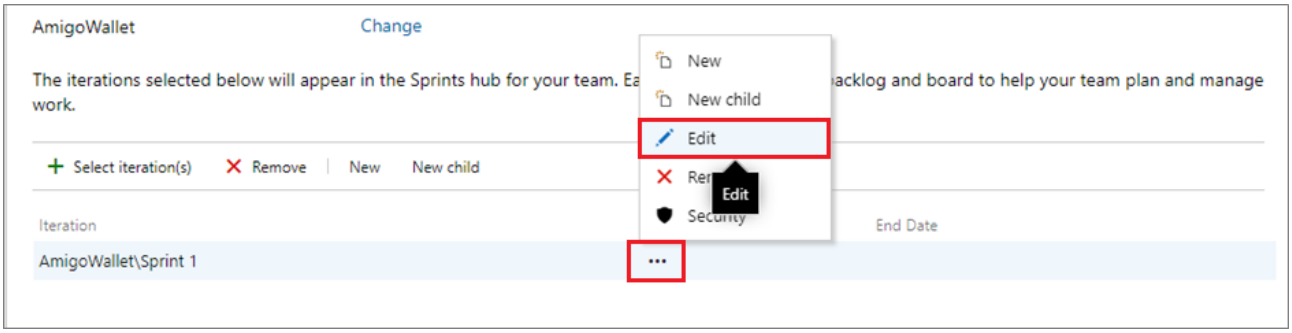


Step 12:

Select AmigoWallet\Sprint 1

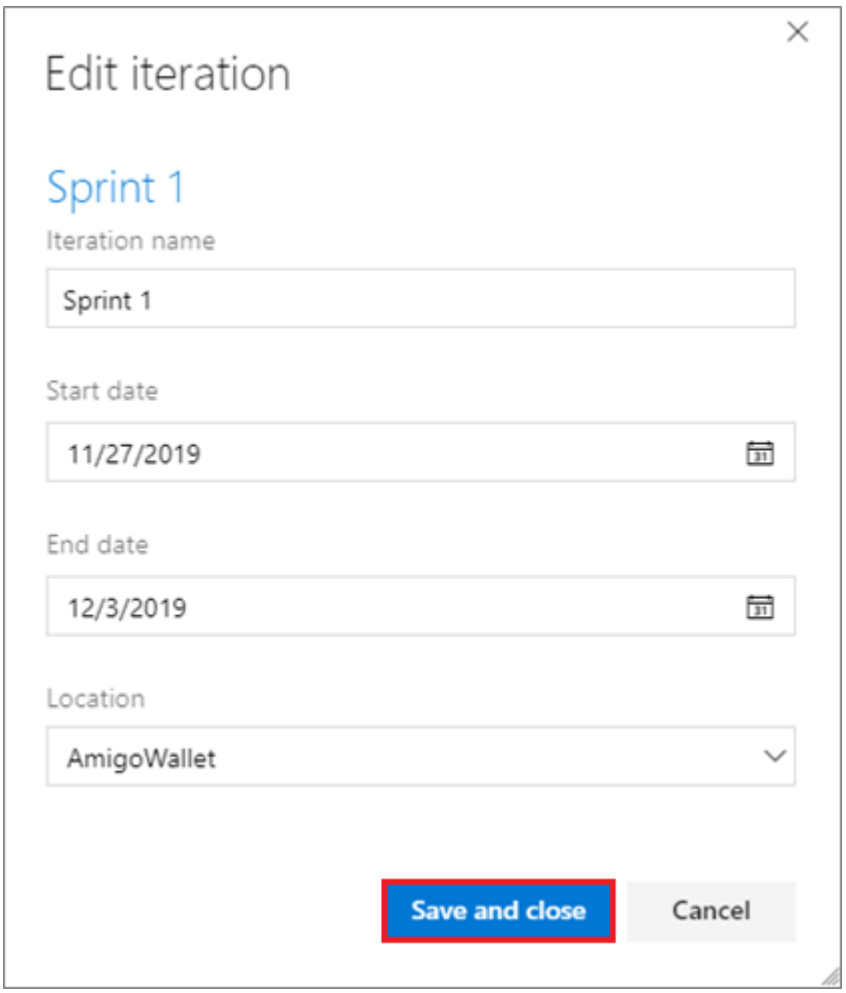
Step 13:

Select the ••• actions icon under **AmigoWallet\Sprint 1** and click on **Edit**:



Step 14:

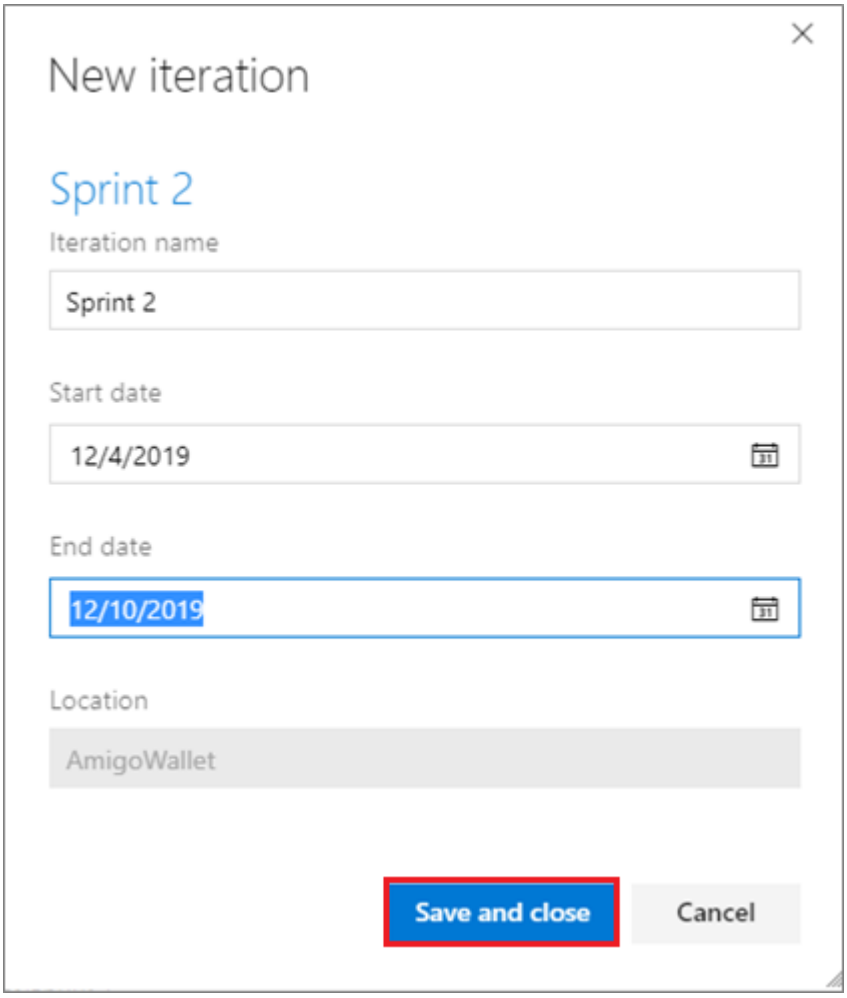
You get a page to edit the iteration (Sprint 1). Fields include the **iteration name, start date, end date** and **location** of the project. **Save and close** as shown:



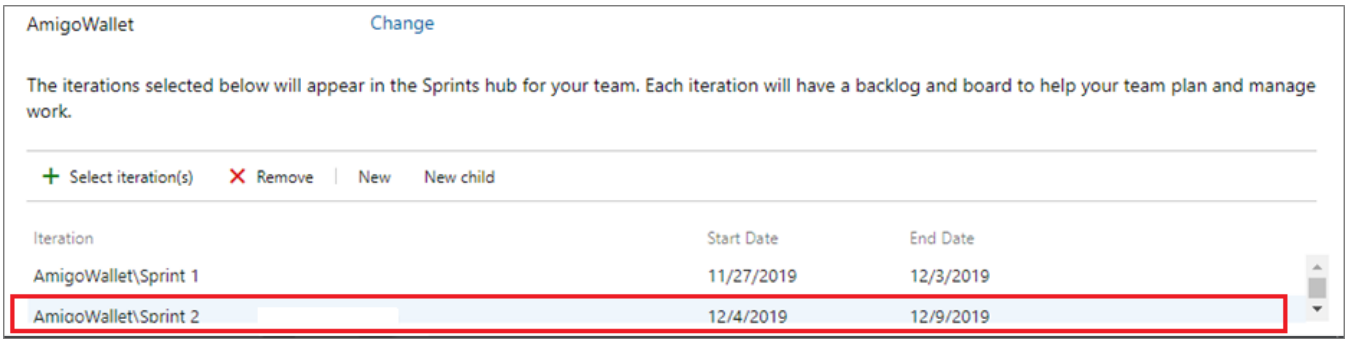
Step 15:

Click on **new** to add new sprint:

Provide values for **iteration name, start date, end date** and **location** of the project for Sprint 2.
Click **Save and close.**



Observe that both the sprints are added successfully.

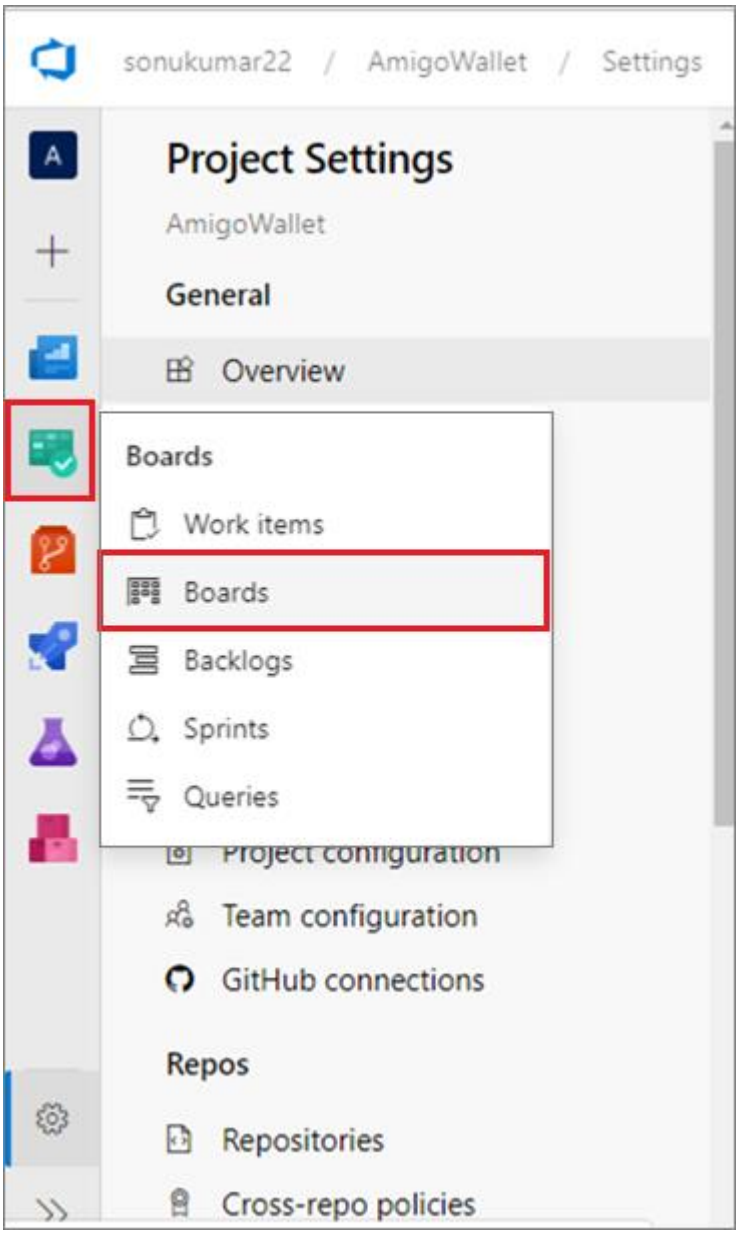


Working with Boards:

Step 1:

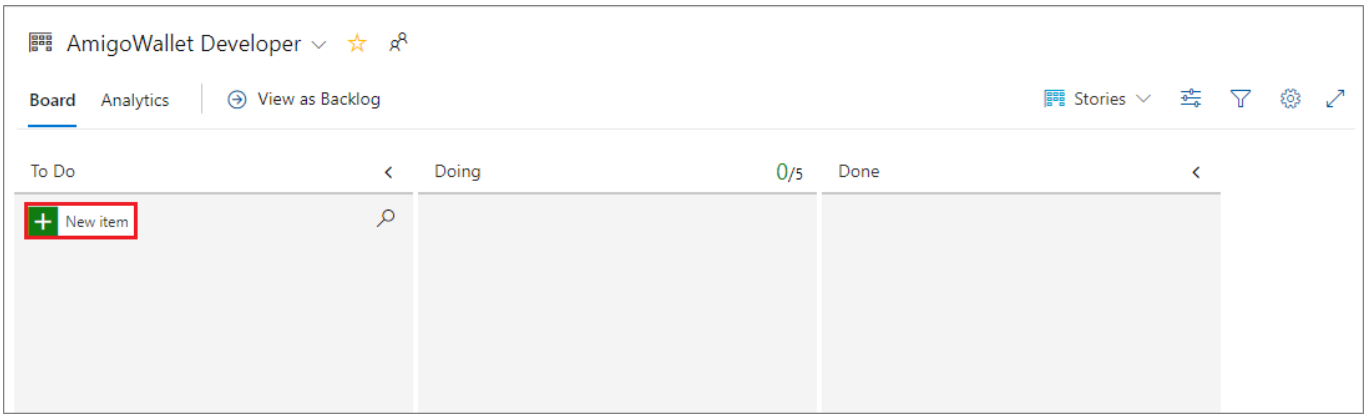
Under Project Settings, select Boards view to understand agile process.

The User Stories Kanban board is the best tool for quickly adding user stories and child tasks. To open, choose **Boards -> Boards.**

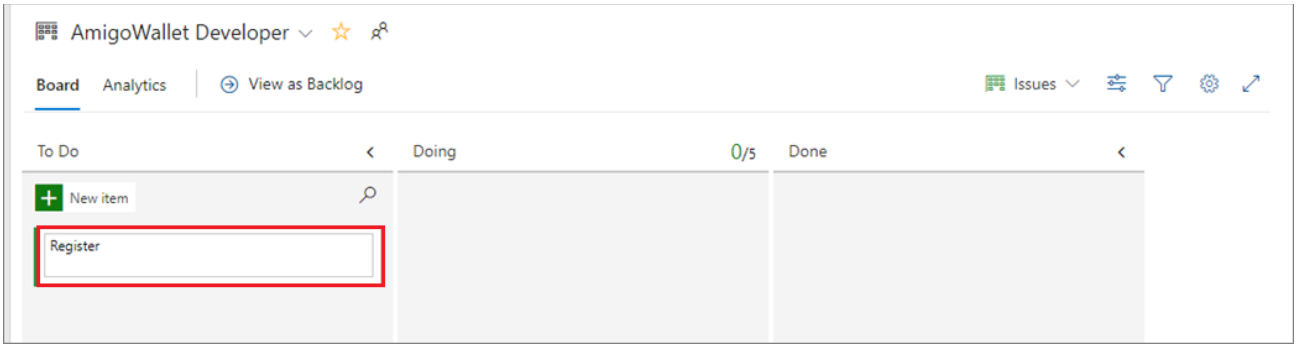


Step 2:

From the User Stories board, choose **New item** and start adding those stories you want to track.

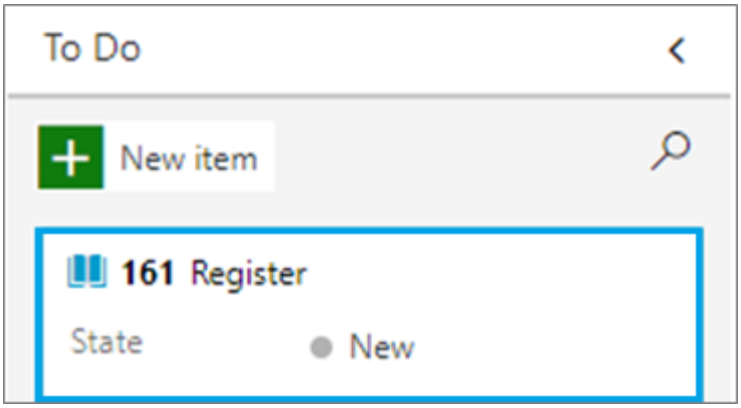


Add Register as the new user story.



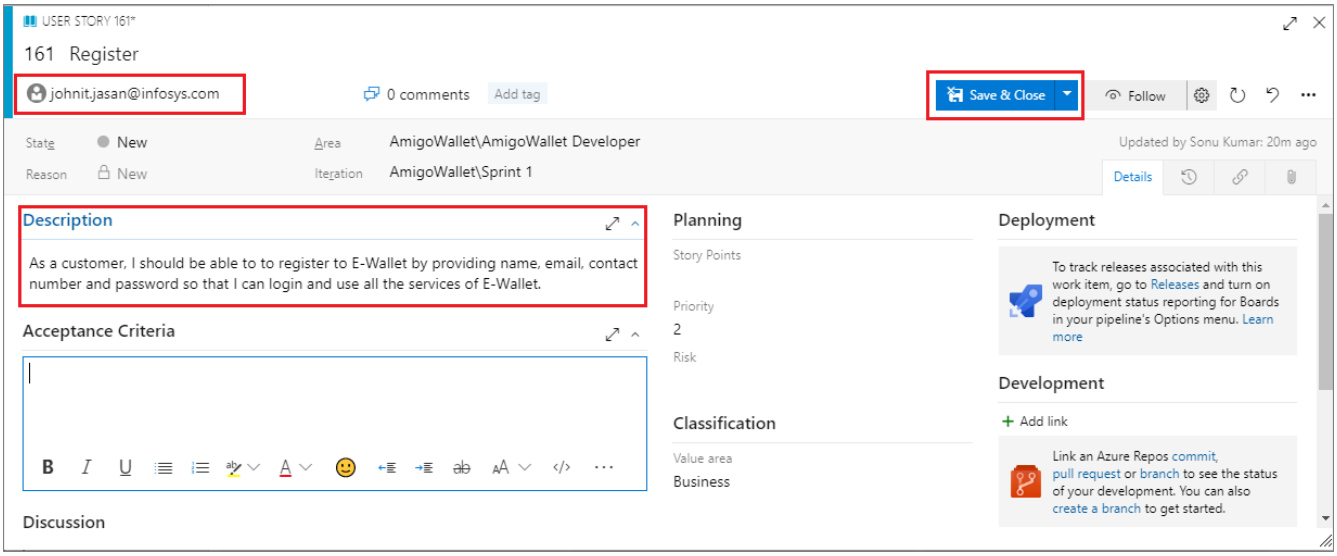
Step 3:

You can observe that the card for Register user story with work id assigned as 161 is created. The state of the user story is new by default.



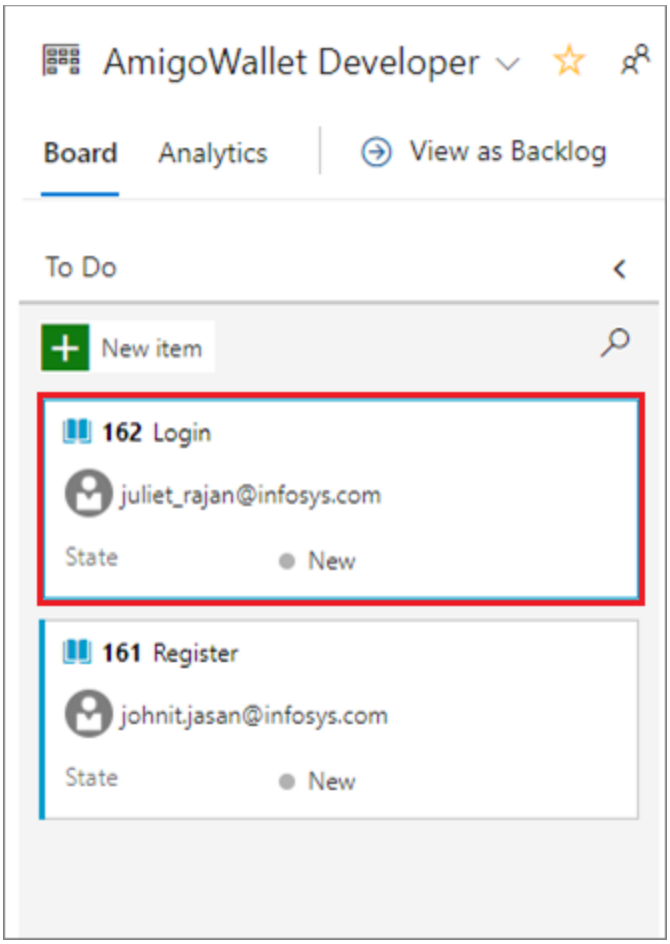
Step 4:

Click on Register user story, to add required details. Click **Save and Close**:



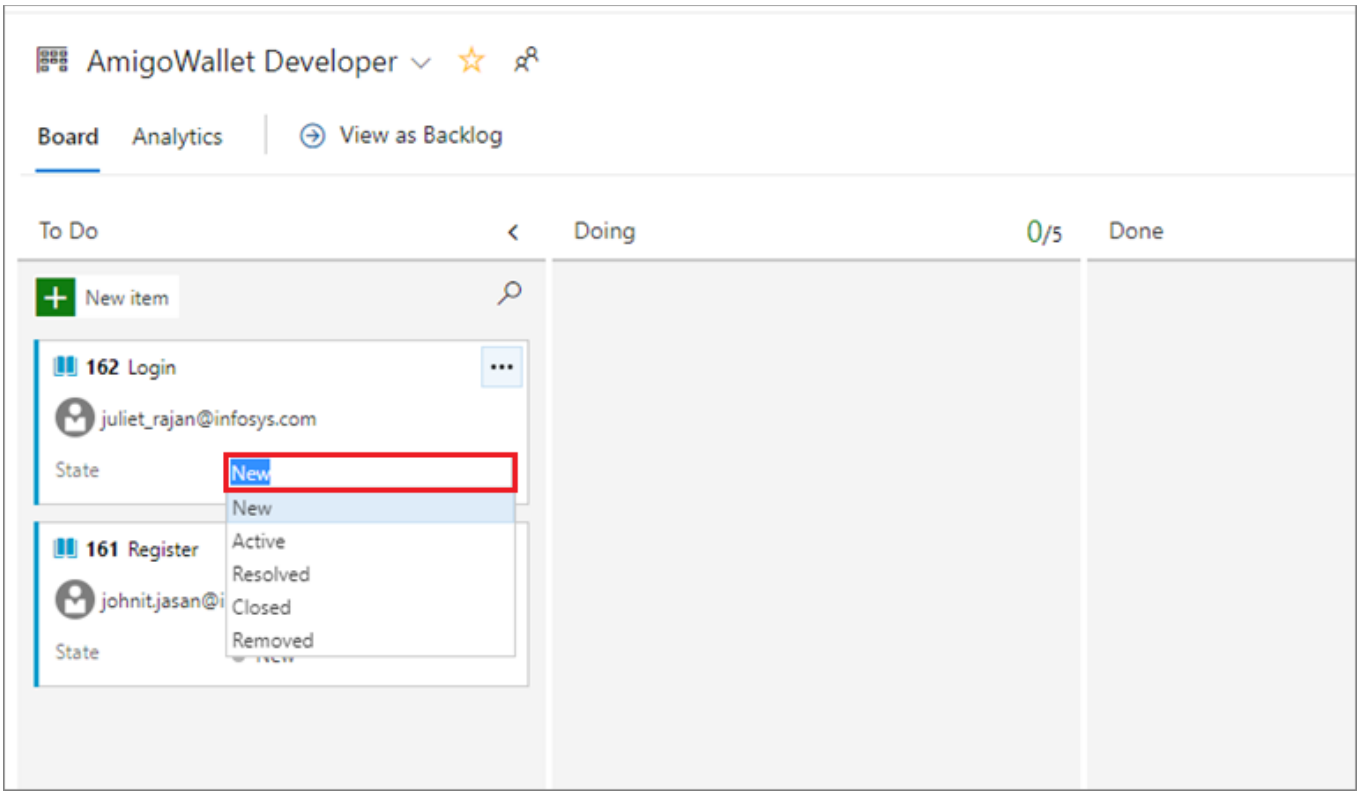
Step 5:

To track the work, you want to manage, add as many user stories that you need. Login user story is added as shown:



Step 6:

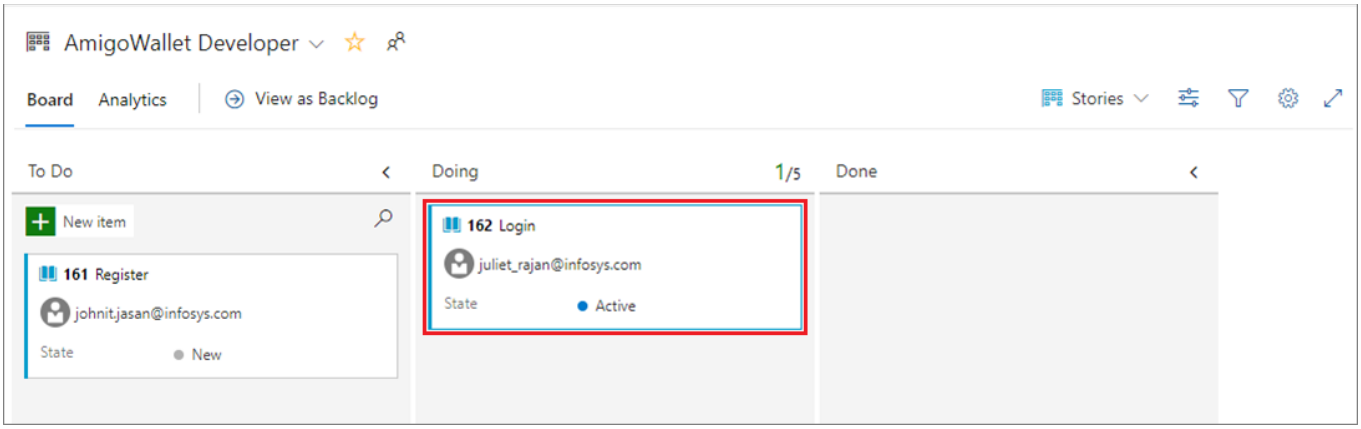
In user story card you can select the state as new, active, resolved, closed or removed:



Based on the state selected, user story card moves to the suitable lane.

- If the user story's state is new, then it falls under **Todo** panel.
- If the user story's state is Active, then it falls under **Doing** panel.
- If the user story's state is Closed, then it falls under **Done** panel.

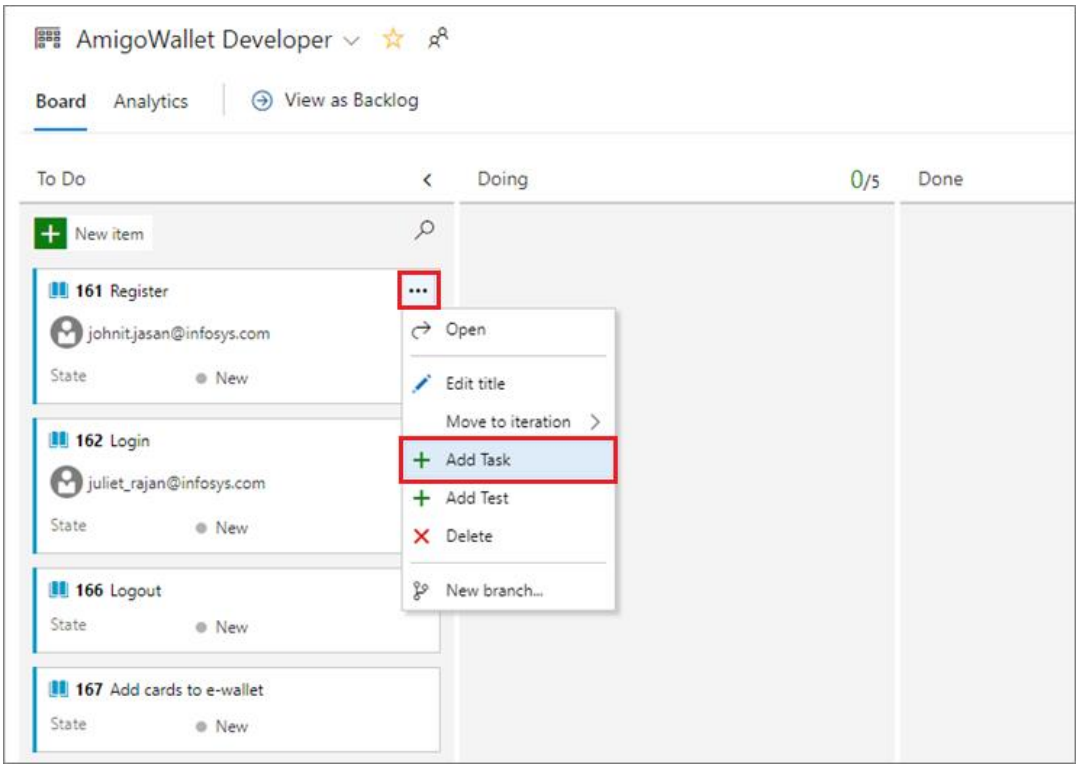
Alternatively, you can also **drag** user story to any panels based on its state.



Step 7:

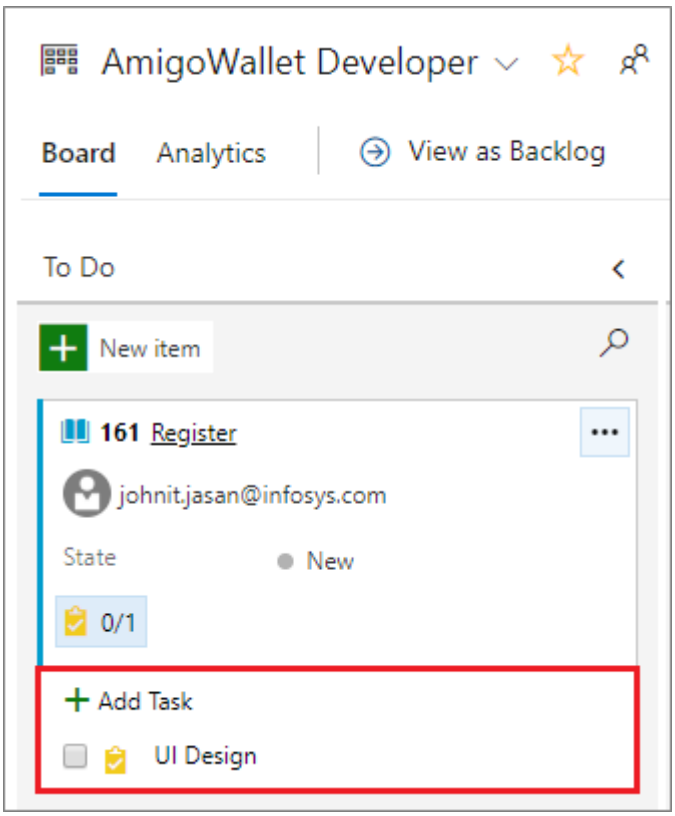
Task checklists provide a quick and easy way to track elements of work which are important to support completing a user story. In addition, you can assign individual tasks to different team members.

To add task for user story, select the ••• actions icon and select **Add task**:



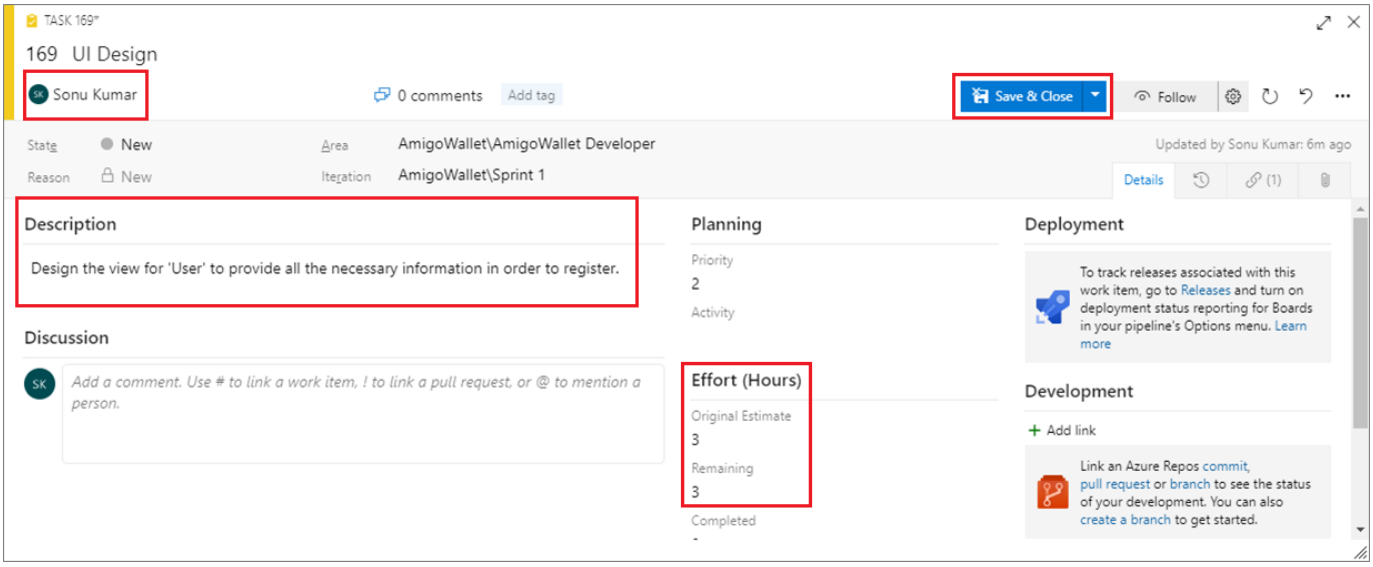
Step 8:

Enter a title for the task as '**UI Design**' and type Enter when done.



Step 9:

You can observe UI Design is added as new **Task**. **Select** newly created Task to see the detailed view.



If you have a number of tasks to add, simply keep typing your task titles and type Enter.

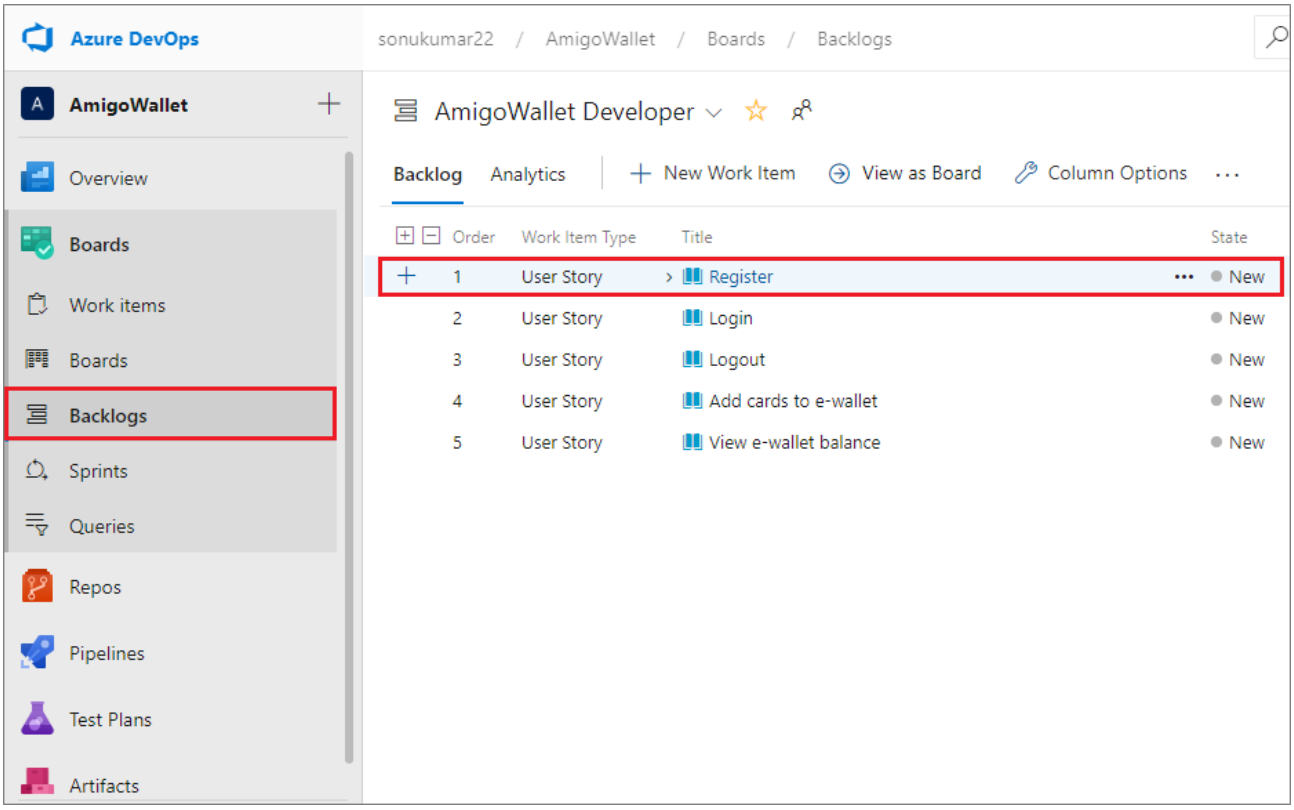
Thus we have seen working with Boards. Since, this is an integrated agile work flow you will keep navigating here.

Backlogs:

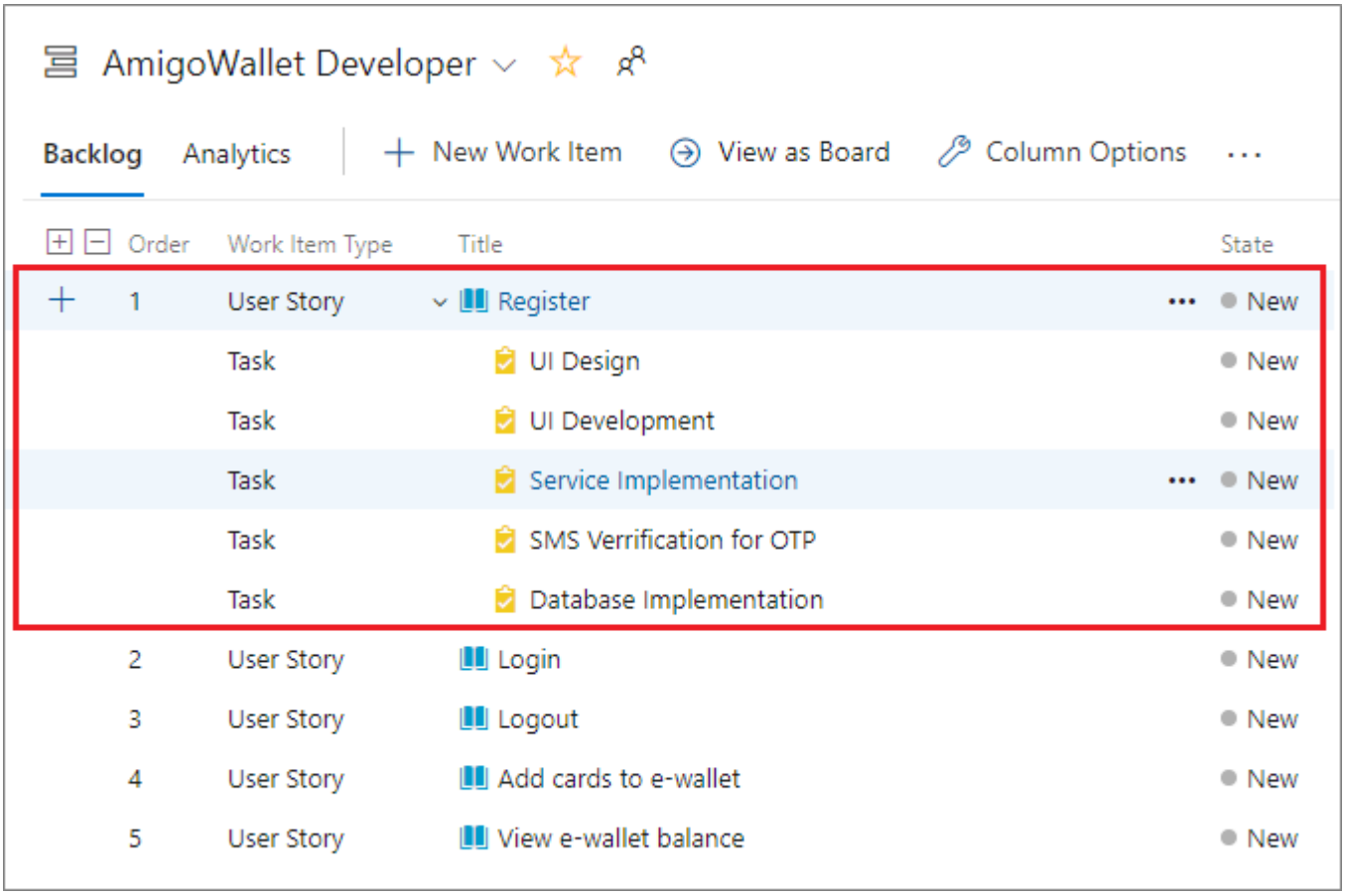
Step 10:

With **Backlogs**, you can quickly plan your project by adding user stories or requirements to your product backlog. Once you have your plan in place, you can start driving code development efforts.

Now navigate to **Backlog** section to observe hierarchy of **Work Items** and **Priority.**



You can observe all user stories created are present in **Backlog** section.

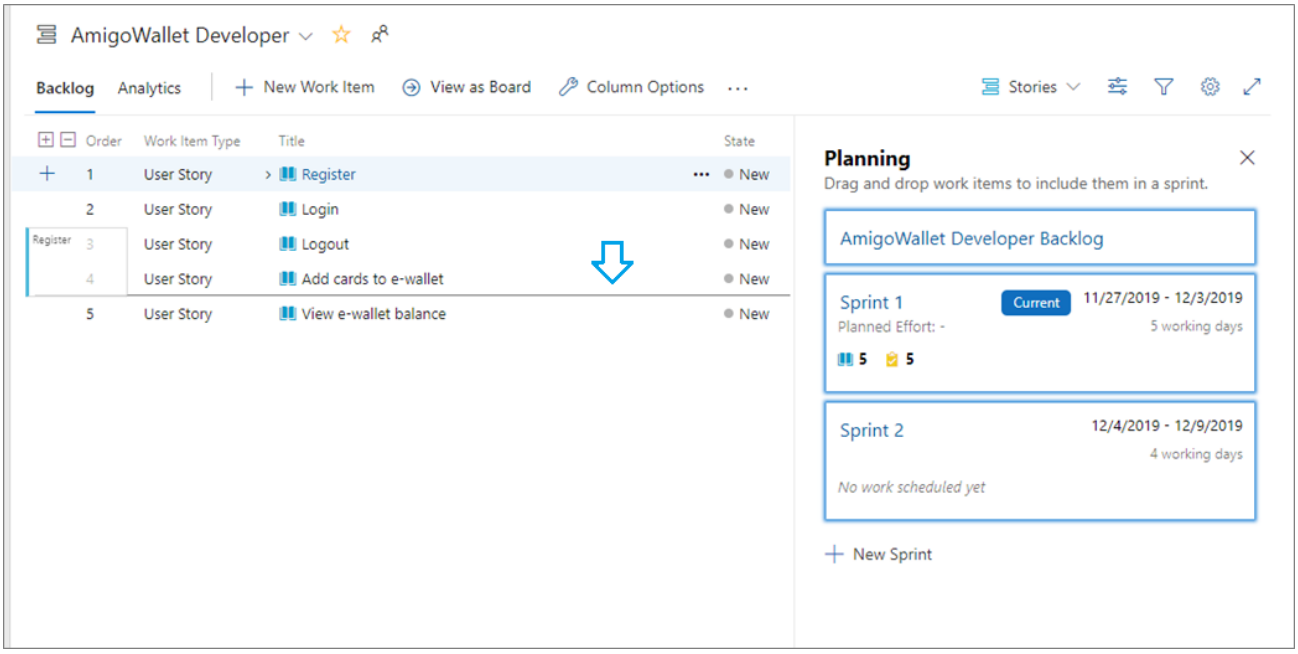


Step 11:

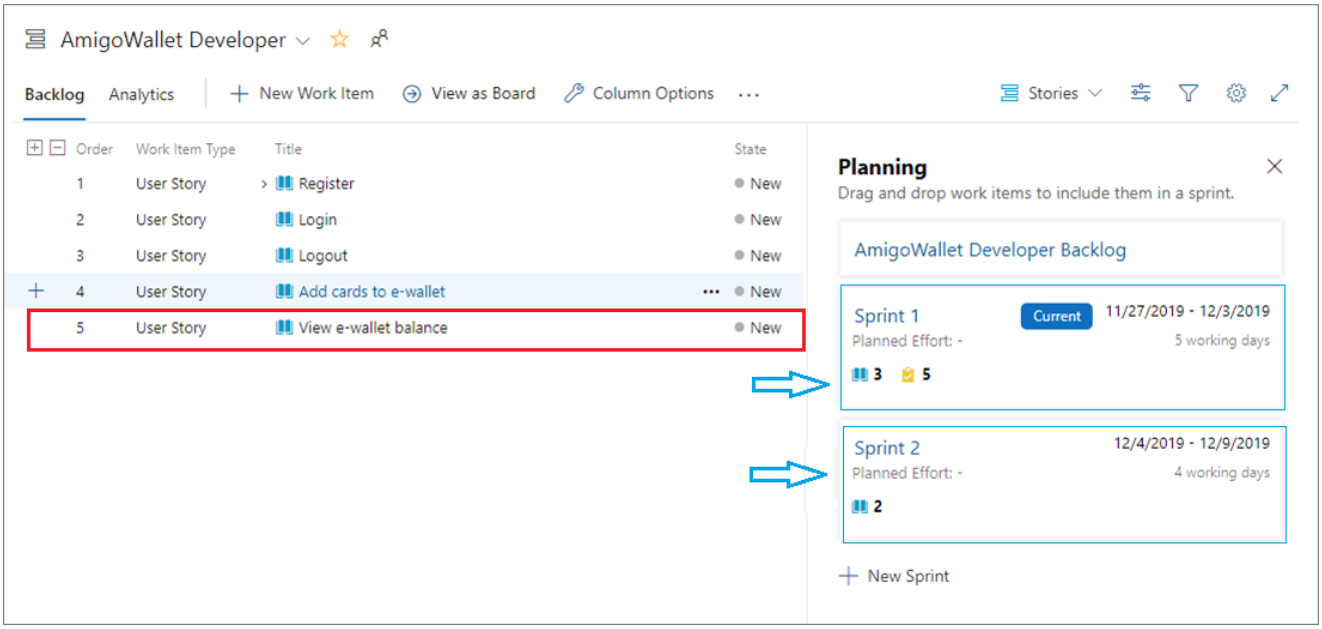
You can reorder them to create a prioritized list of work. Review and prioritize your backlog frequently to help your team know what's most important to deliver next.

To reorder a backlog, you must have Basic or higher level access.

To reorder your backlog, **drag** the **work items**. Or, if you prefer to use the keyboard, hold down the **Alt key** and use the **up and down arrows**.

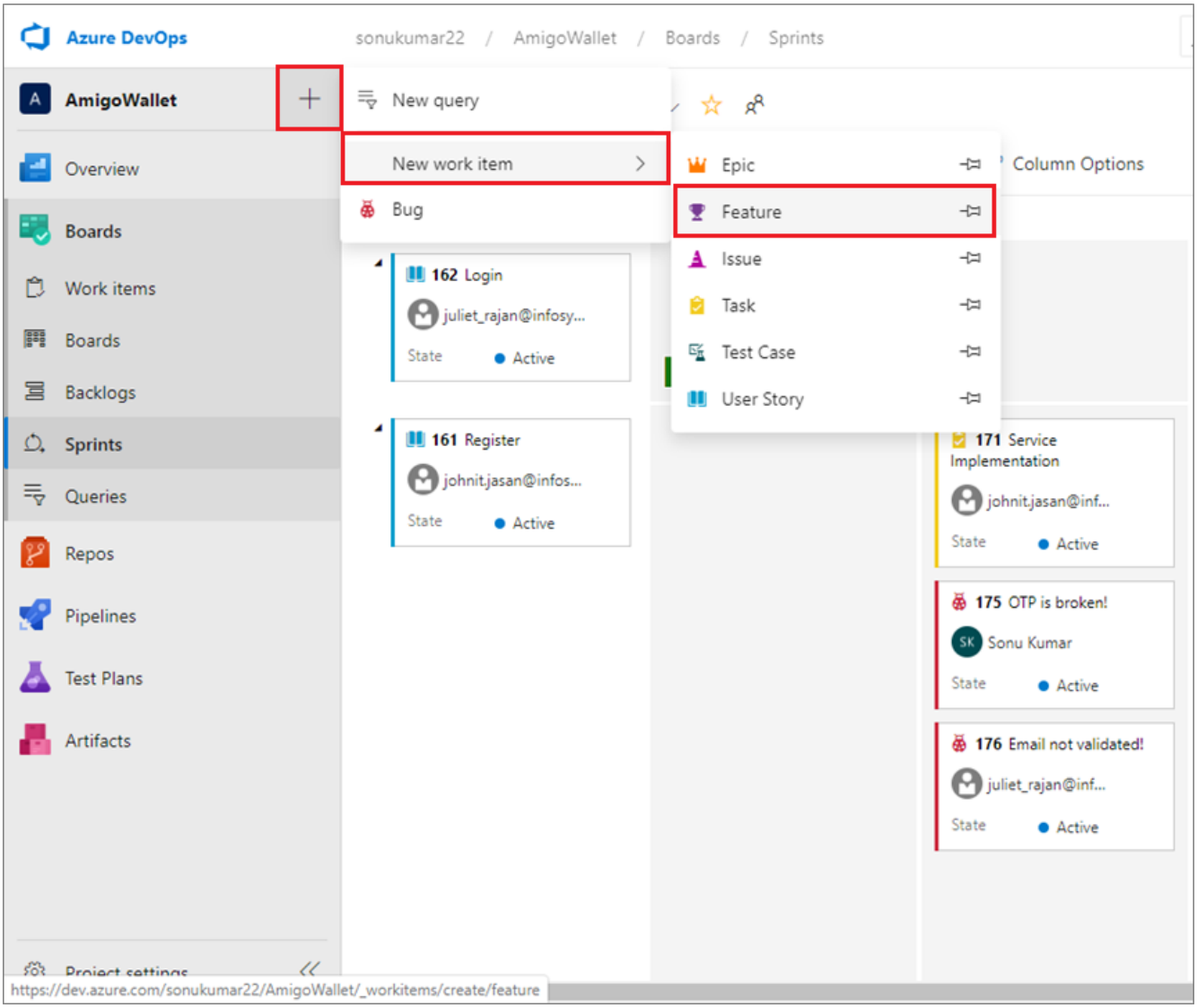


You can also select sprint for the work items. To add a work item to particular sprint, **drag** the **work items** to respective sprint as shown



Step 12:

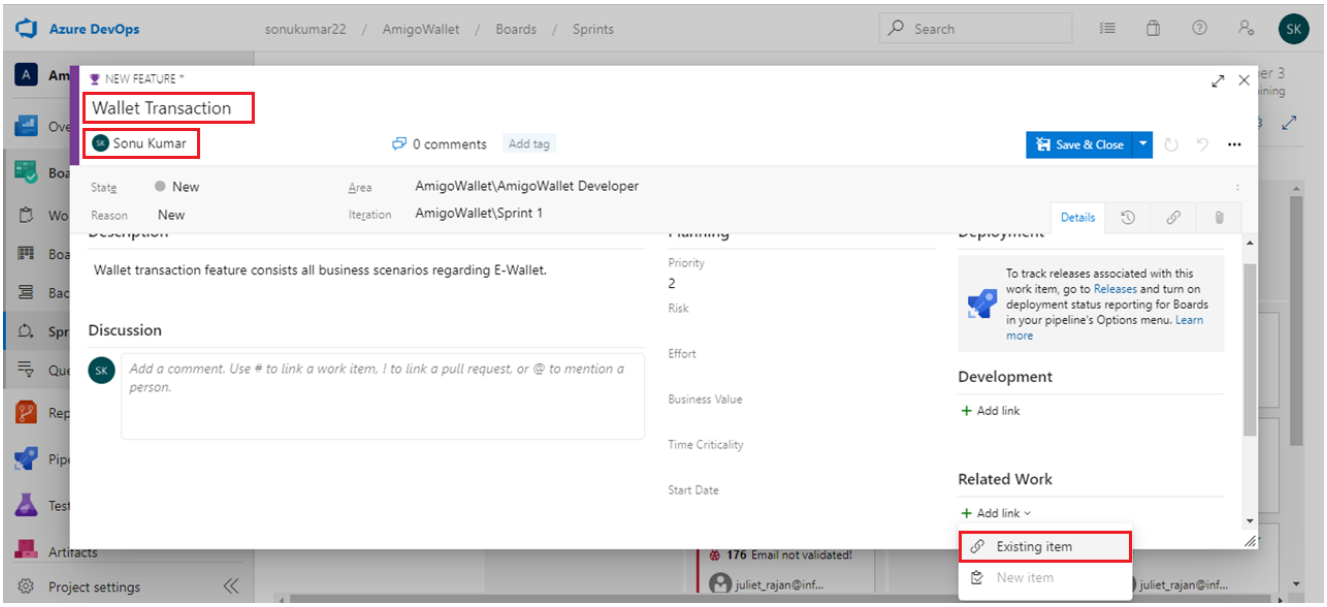
User stories can be logically grouped into **Features.** In order to create a feature **click** + button right next to project name i.e. **AmigoWallet** then **Select New Item > Feature.**



Give feature's **Name** as **Wallet Transaction** and also provide details for **Assigned to, Description** etc.

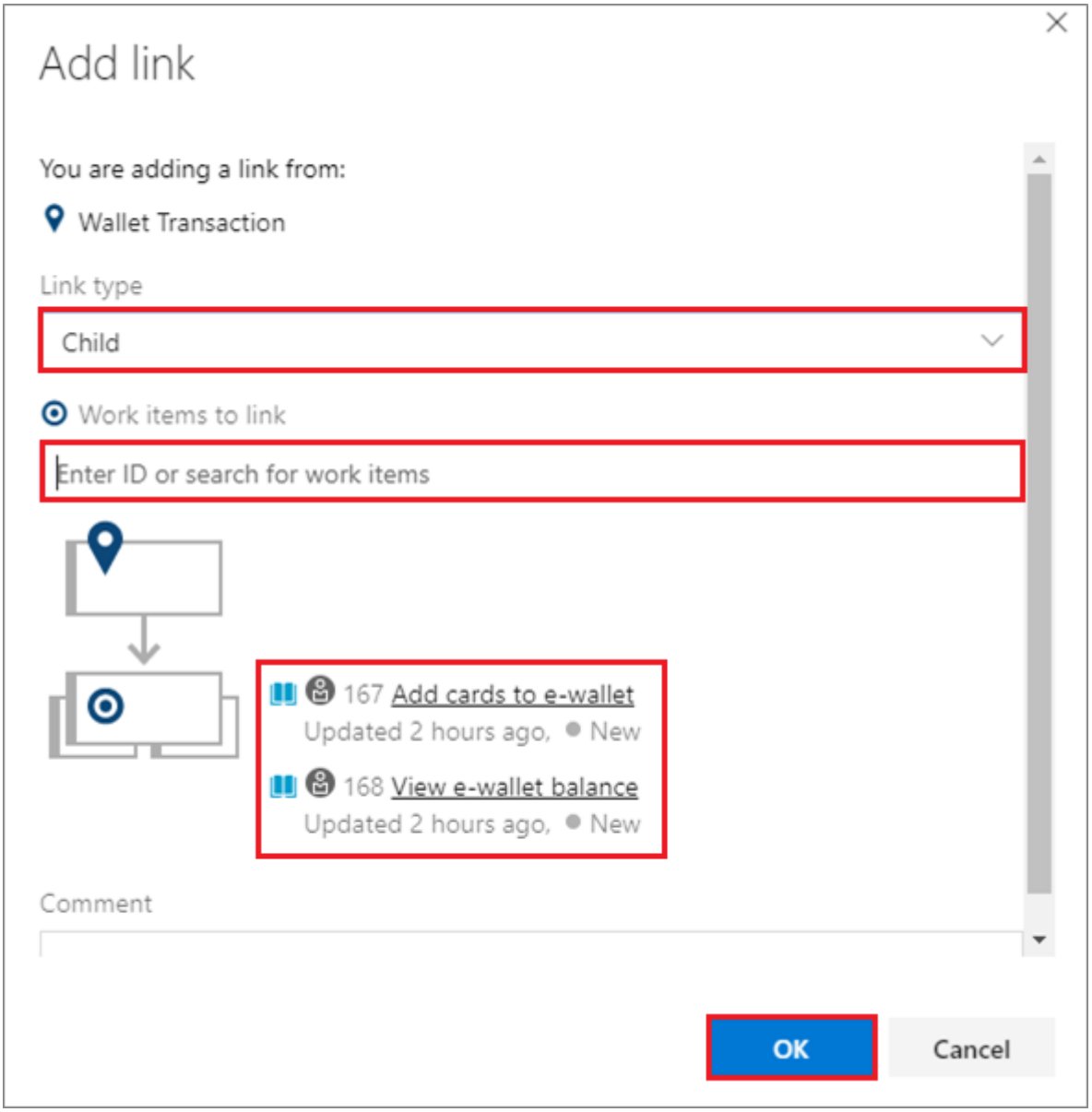
Here we are following bottom up approach meaning **User story** is created prior to **Feature.** So while creating feature you have to establish relation between set of user stories and **Wallet Transaction.**

In order to establish relation, **select Related Work > Add Link > Existing item.**



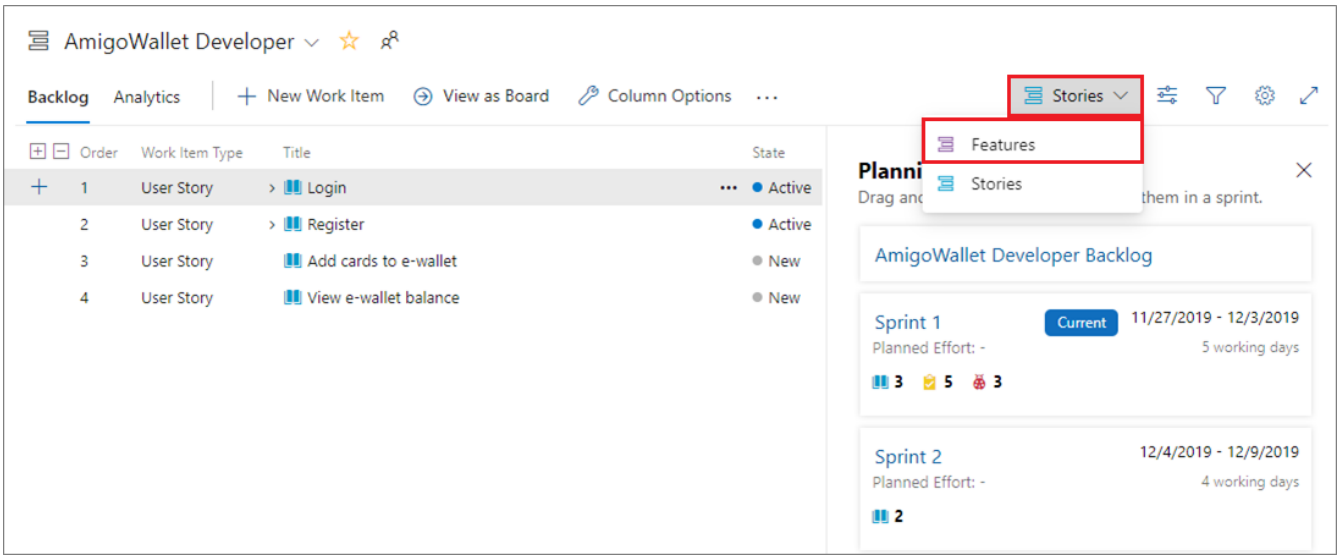
Select **Child** from **Link Type** dropdown and Choose '**Add cards to e-wallet**' and '**View e-wallet balance**' user stories from **Work Items** and **click OK** then **Click Save and Close**

Both user stories will be added as child feature '**Wallet Transaction**'.

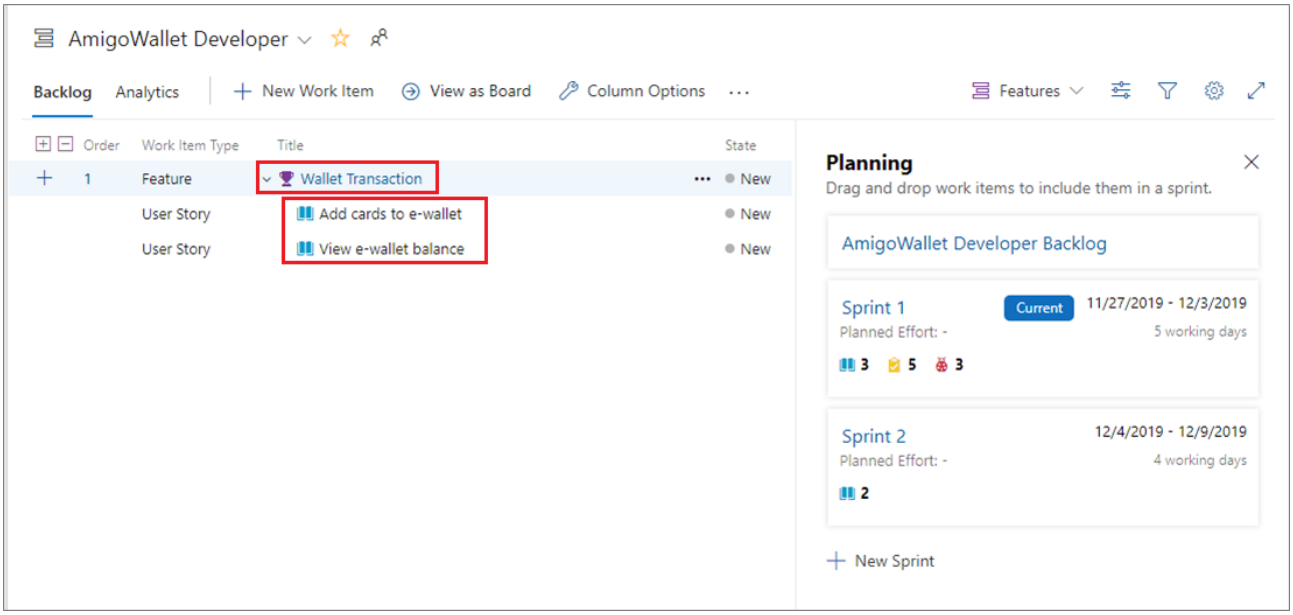


Step 13:

You will be navigated to **Backlog** Section. Select **Feature** instead of **Stories** from dropdown as shown



You will be navigated to feature view.

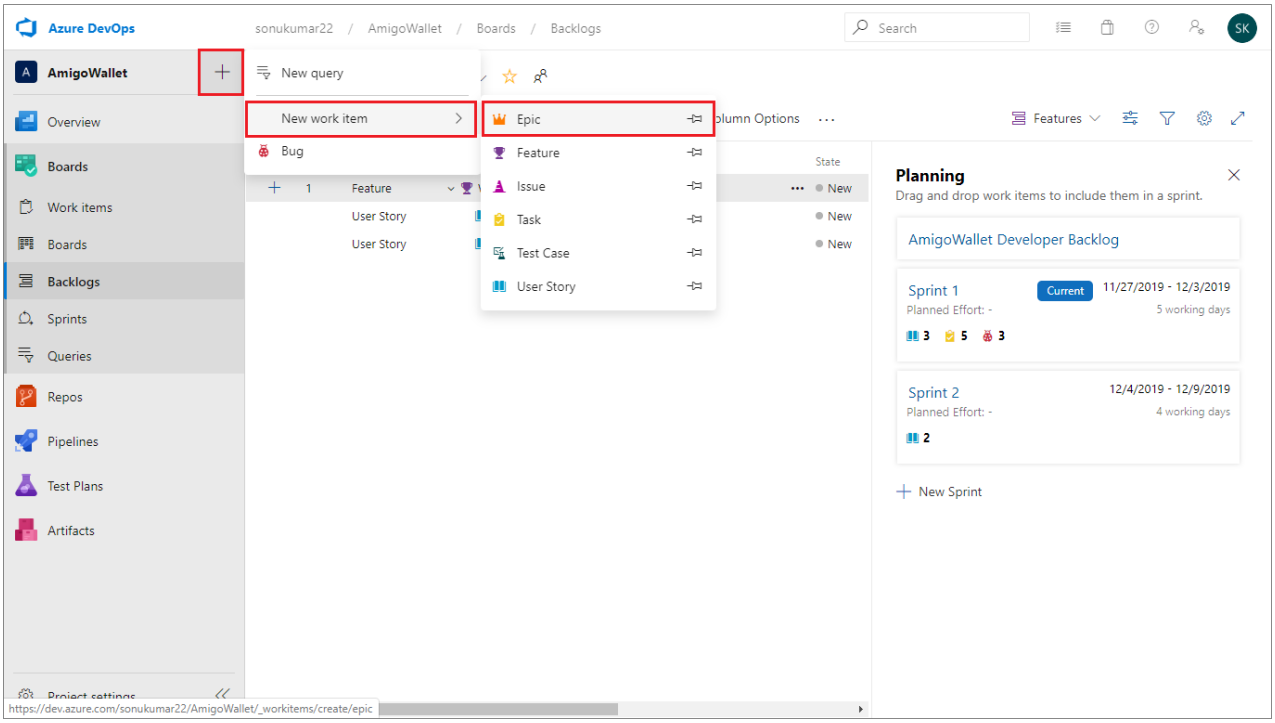


Observe **Wallet Transaction** is added as **Feature** which is **Parent** and inside that '**Add cards to e-wallet**' and '**View e-wallet balance**' user stories are present as **Child.**

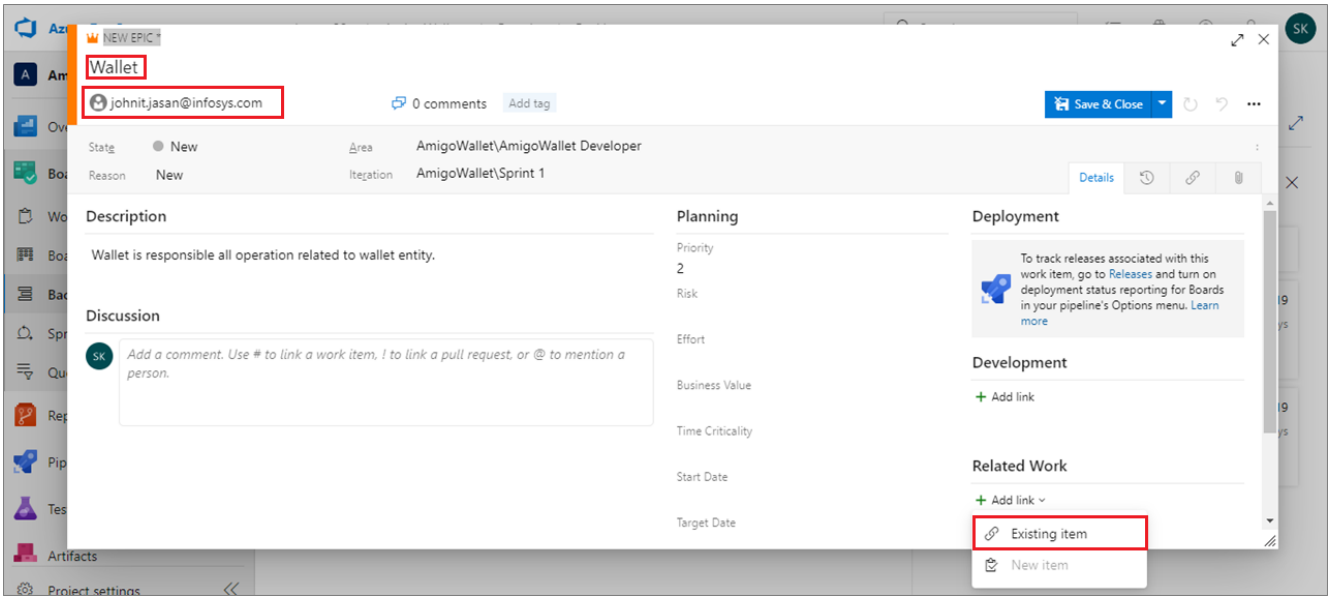
Step 14:

Features can also be logically grouped and collection is said as **Epic**.

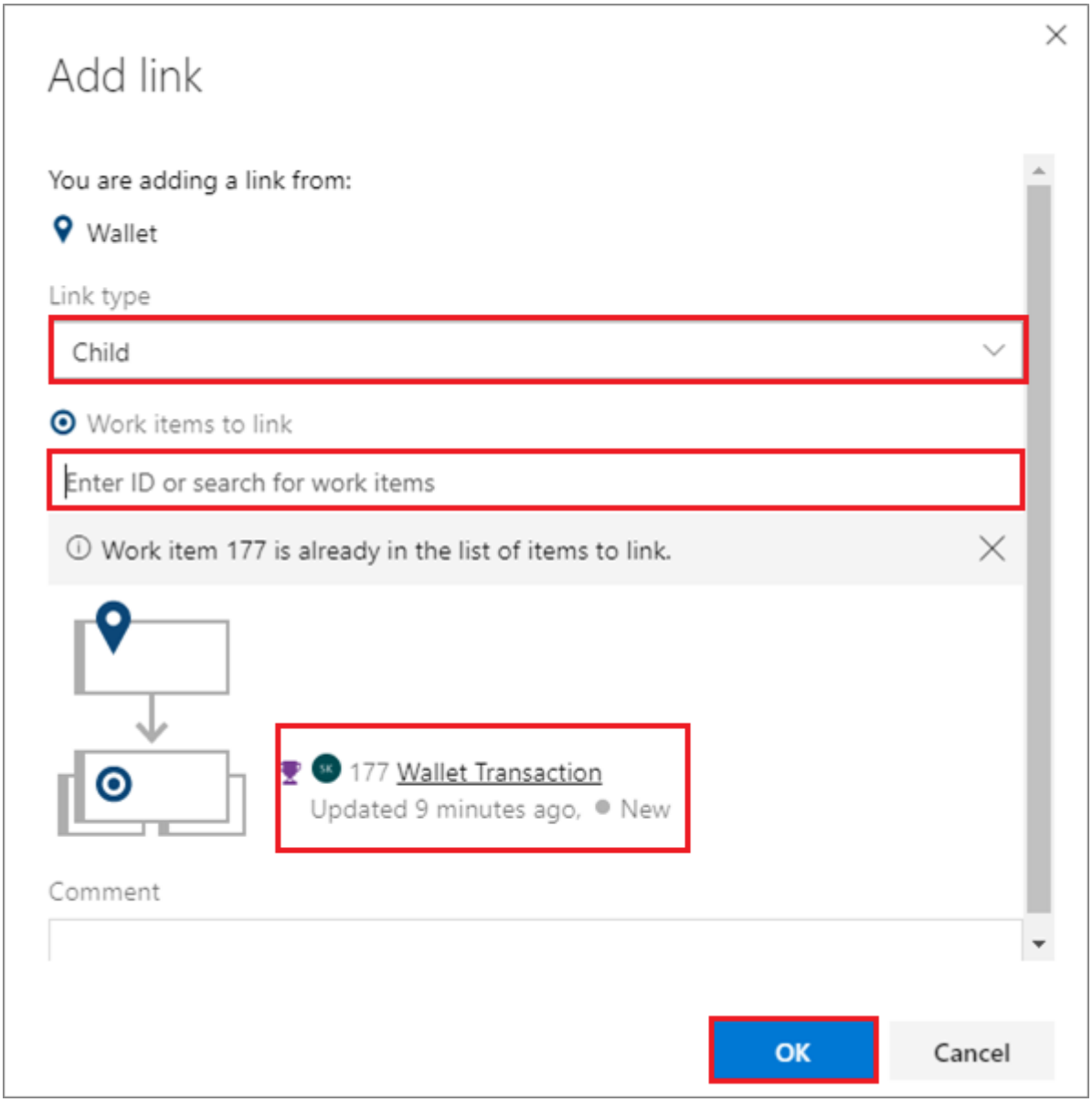
You can add an epic by following same steps as **creating features.** This time **Select New Item > Epic.**



Give this epic **Name** as '**Wallet**' and provide other information. For establishing relation between '**Wallet'** epic and '**Wallet Transaction**' feature **select Related Work > Add Link > Existing item.**

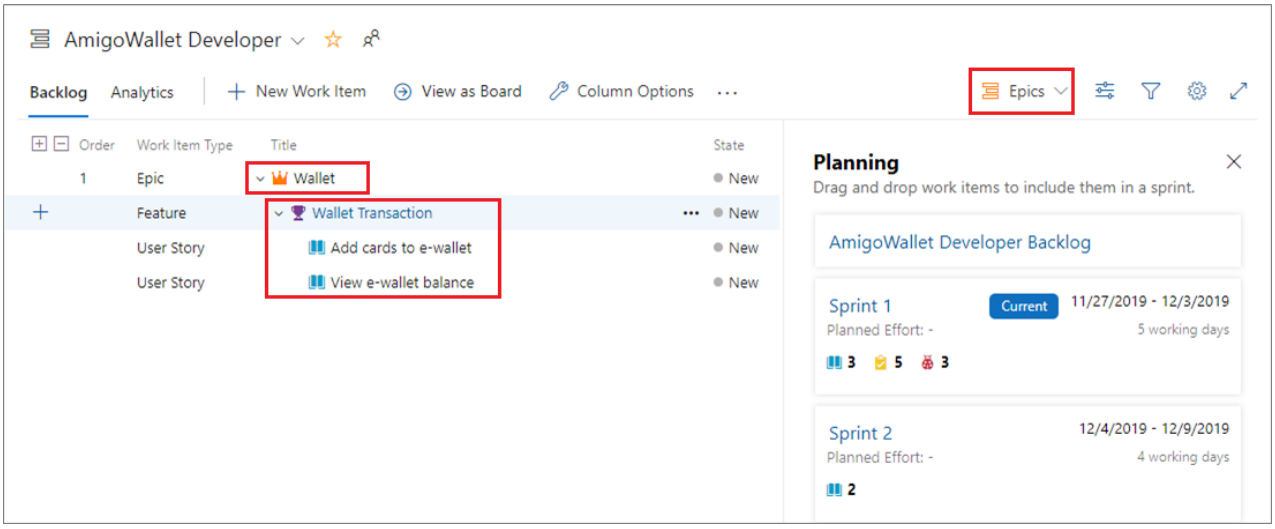


Select **Child** from **Link Type** dropdown and '**Wallet Transaction**' feature from **Work Items** and **click OK** then **Click Save and Close**



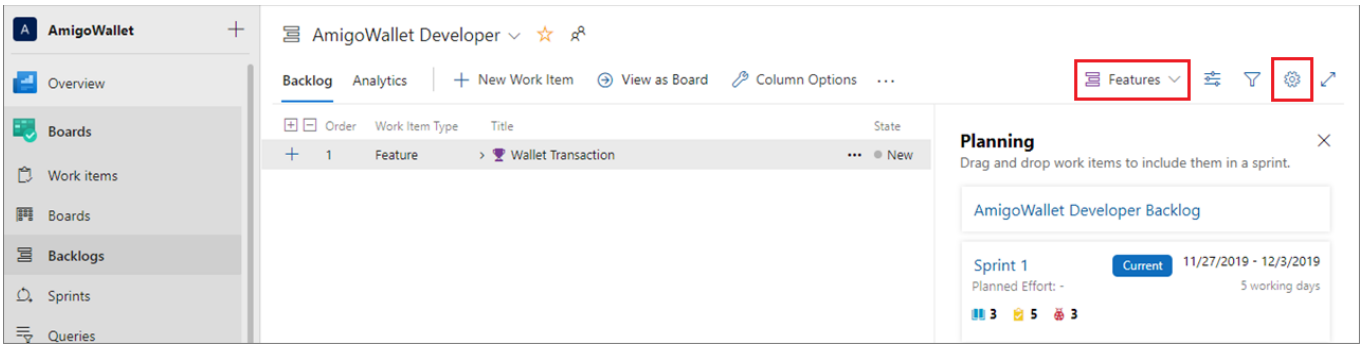
Step 15:

You will be navigated to **Backlog** Section. Select **Epic** instead of **Stories** from dropdown as shown

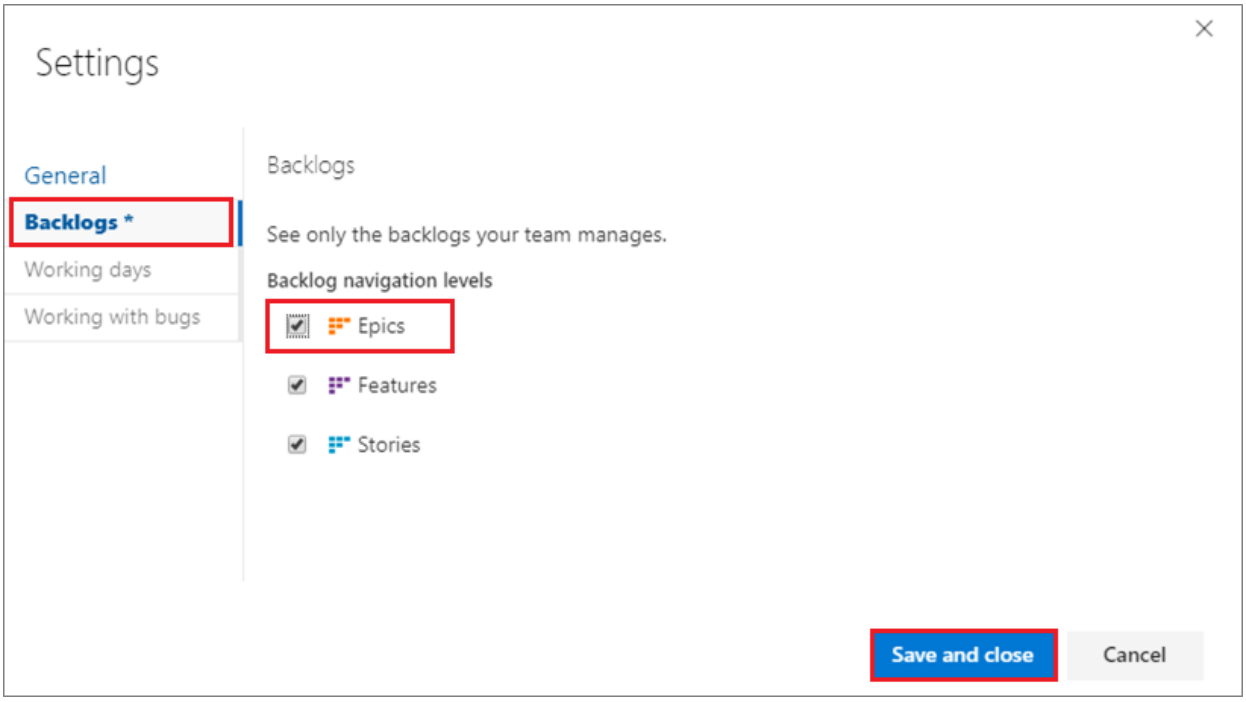


You can observe '**Wallet'** is added as **Epic** and '**Wallet Transaction**' **Feature** as child of it.

If in dropdown you are not getting **Epics** as an option, then **select Settings.**



Now **Select Backlogs** and **check Epics checkbox**.



Now you will be getting **Epics** in dropdown.

Sprints:

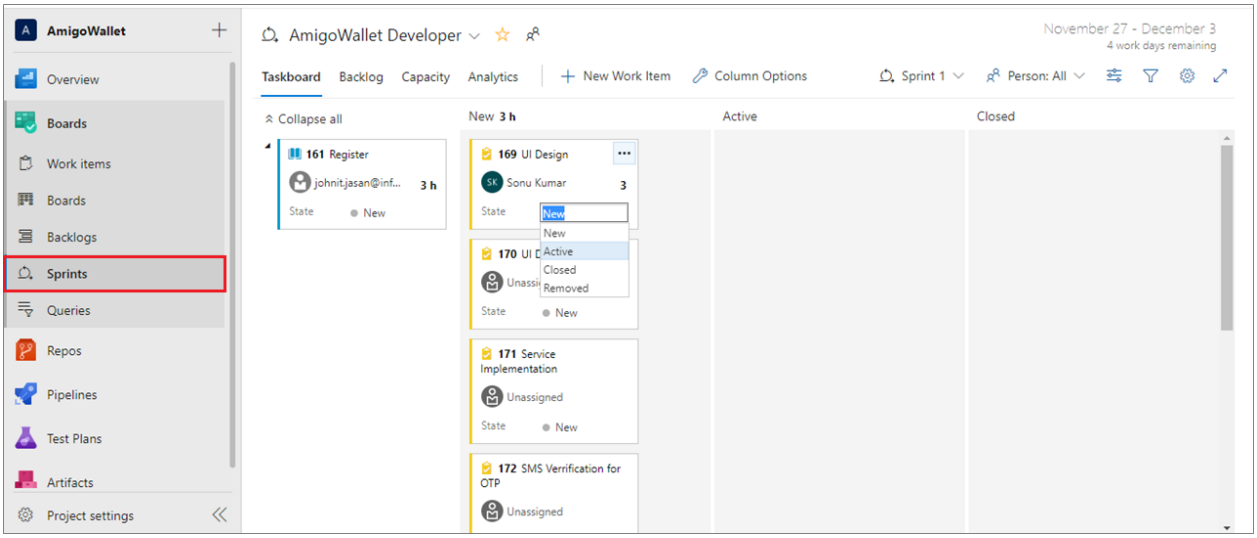
Step 16:

As you have seen, from your **backlog**, you can map work to an iteration path using drag-and-drop.

You can view that work in a separate **sprint backlog**.

Navigate to **Sprints.**

Sprint **backlogs** and **taskboards** provide a filtered view of work items a team has assigned to a specific iteration path, or sprint. Sprints are defined for a project and then selected by teams.

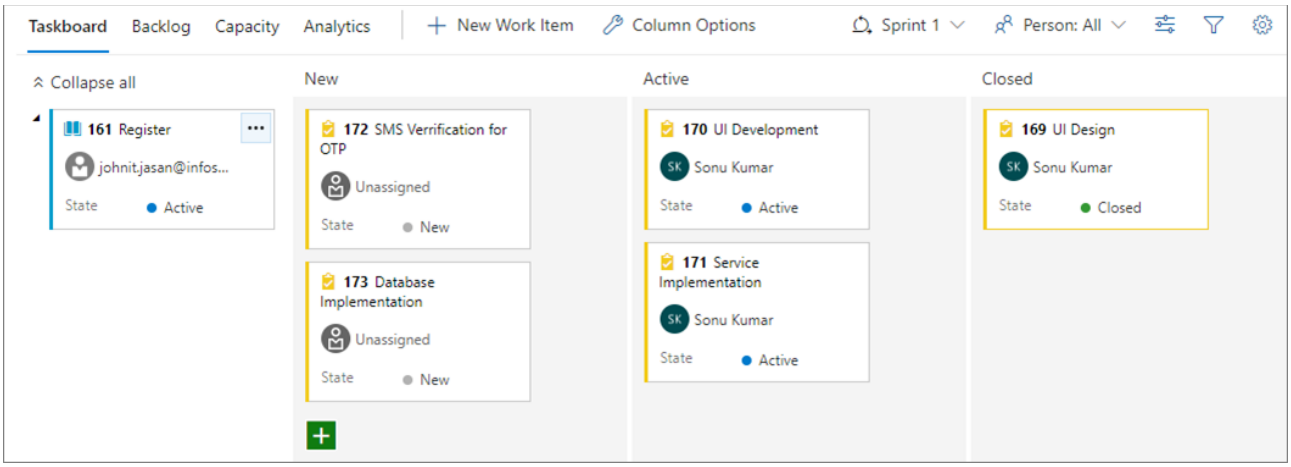


Step 17:

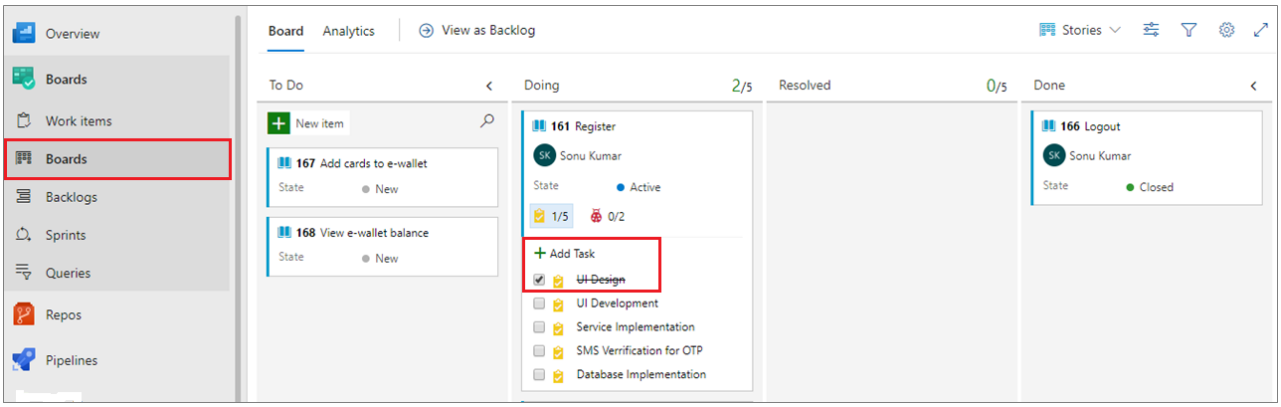
Observe that in **taskboard,** every task that is added in '**Register**' user story is present in **New** lane.

You can change **state** of task based on it status to move in **Active** or **Closed** lane or drag and drop to respective lanes.

Set state of '**UI Design**' to **Closed** and '**UI Development**' and '**Service Implementation**' to **Active** state.



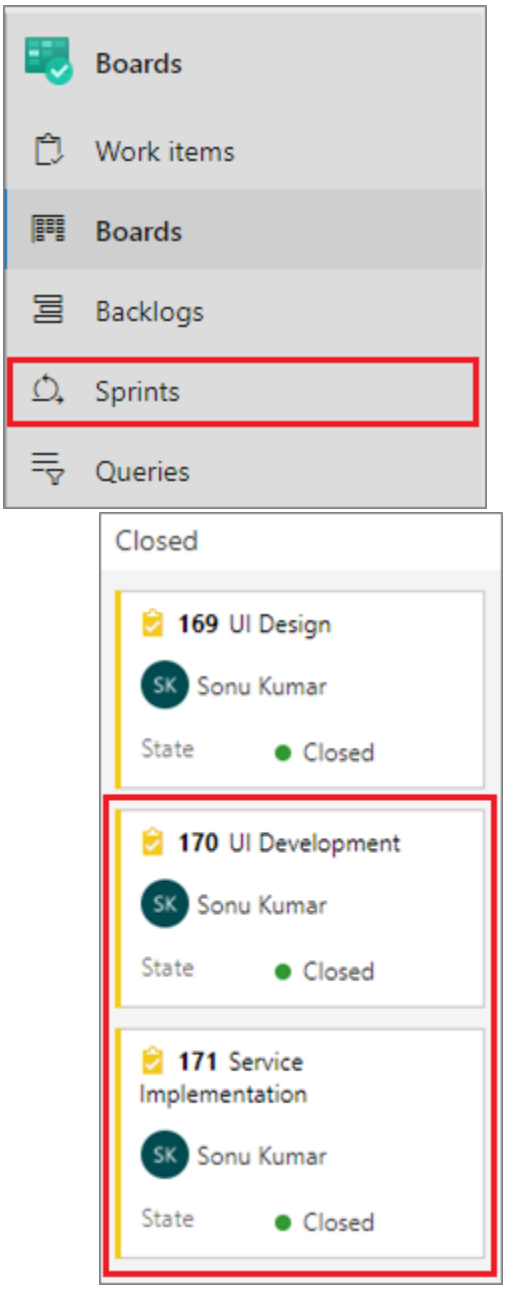
As you have closed '**UI Design**', Navigate to **Boards.**



Step 18:

You can observe that '**UI Design**' task under '**Register**' user story is **checked** i.e. completed.

Now let's try to marks '**UI Development**' and '**Service Implementation**' task as **checked** here and Navigate to **Sprints.**

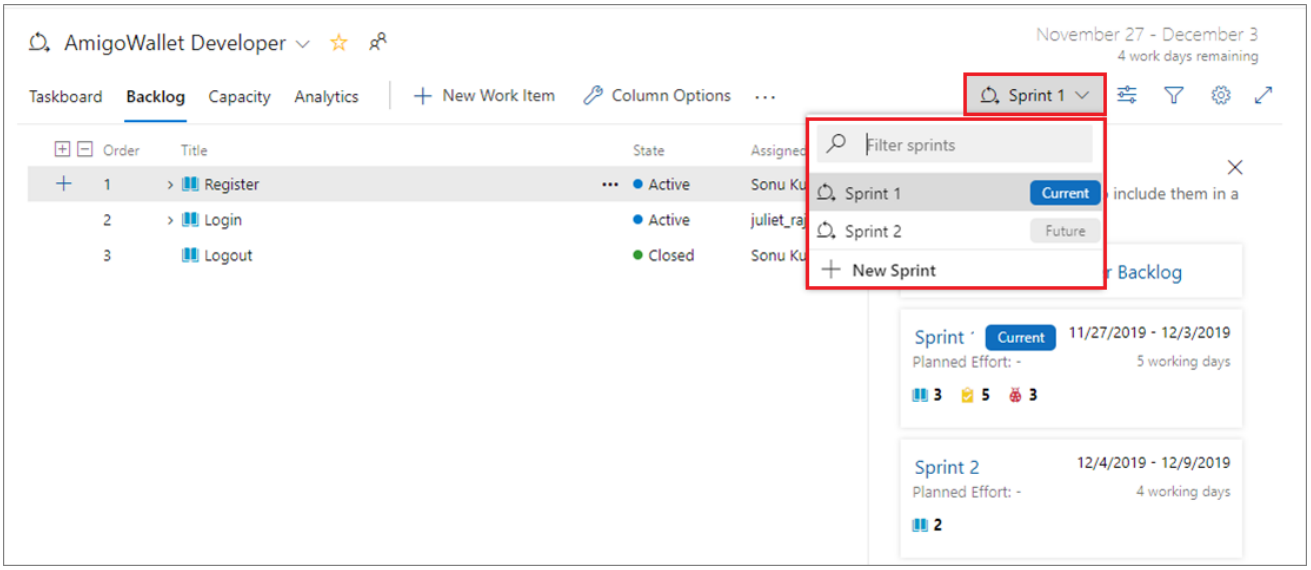


Observe that both '**UI Development**' and '**Service Implementation**' tasks are in **closed** state.

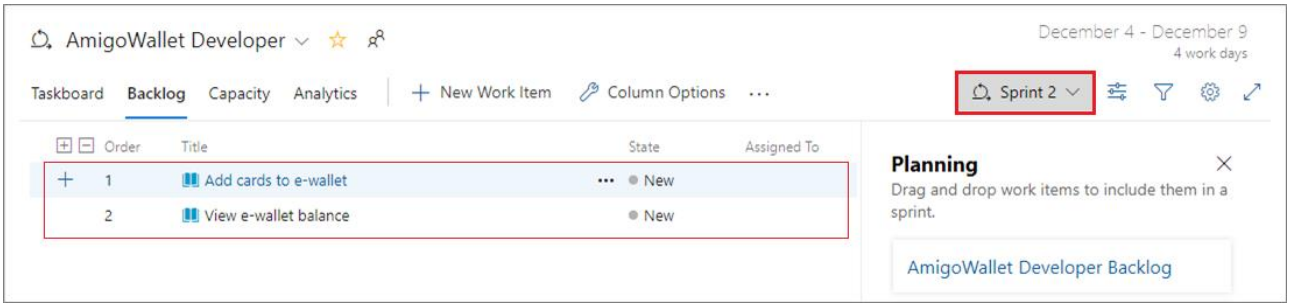
So, you can even close the tasks or user stories from **Boards**.

Step 19:

You can see work item that are part of different sprints as well. Select **Sprint2** from dropdown



Now work items related to Sprint2 will be listed here.



Swimlanes and Columns:

Your Kanban board supports your ability to visualize the flow of work as it moves from just defined to completed. When you add swimlanes, you can also visualize the status of work that supports different service-level classes. You can create a swimlane to represent any other dimension that supports your tracking needs.

For example, you can create three swimlanes—Expedite, Standard, and Parked—to track high-priority work, standard work, and work that's currently blocked.

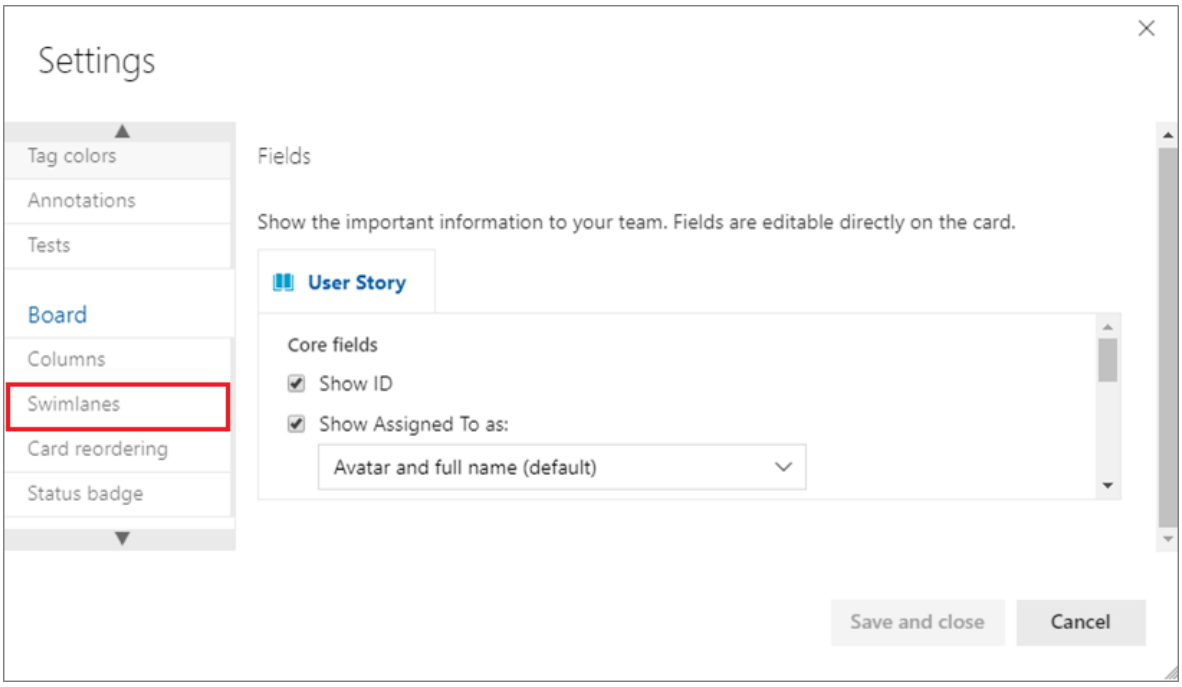
You will create Expedite swimlane in this demo and add 'Register' inside Expedite to make this high priority.

Step 20:

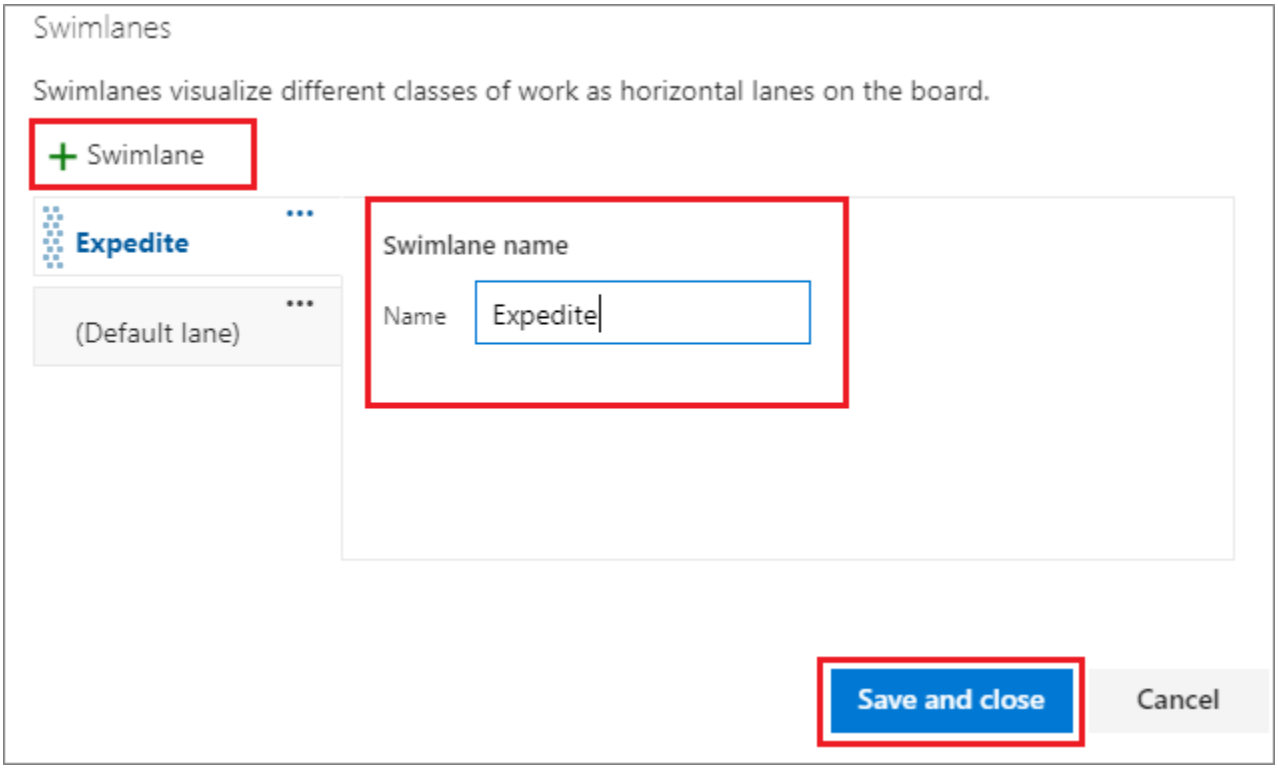
Navigate to **Boards. Select Settings.**

Step 21:

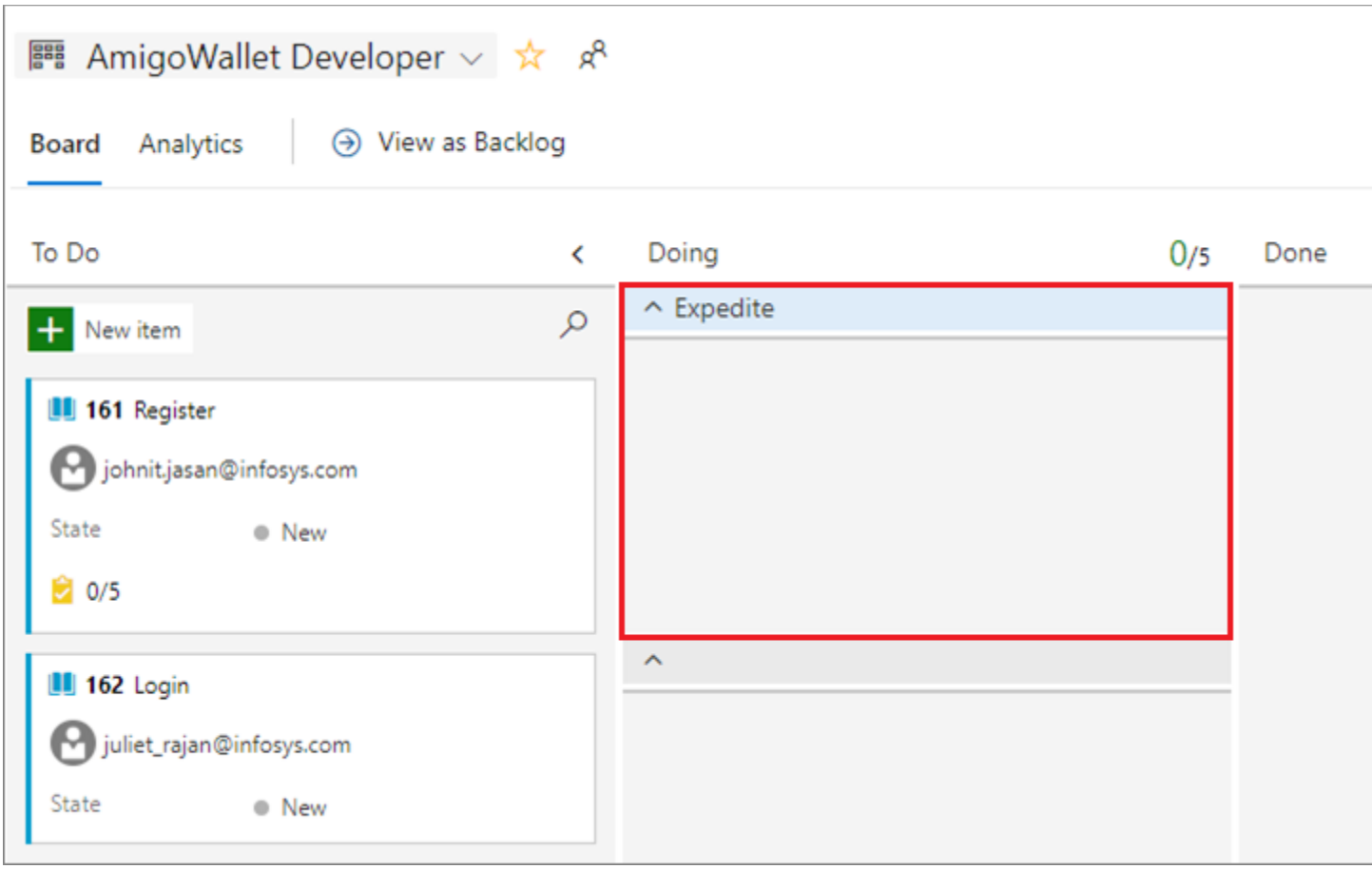
Select Board > Swimlanes



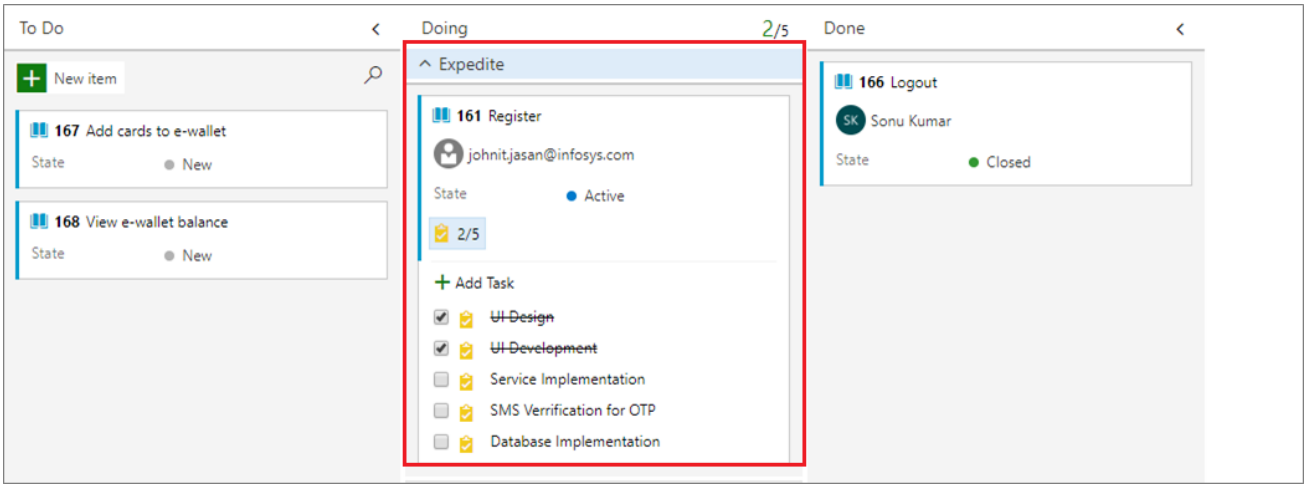
Select **Swimlane** and give it name as **Expedite** and **Click Save and Close**.



Navigate to **Boards.**



Observe **Expedite** lane is added horizontal to **Doing** lane. Drag **Register** to **Expedite** lane.



If you want to delete **Expedite** lane, make sure first remove all work items from it and then **Delete**.

You can even add different columns for Kanban board based on requirement. Here we have requirement that all work item before getting closed should be reviewed once.

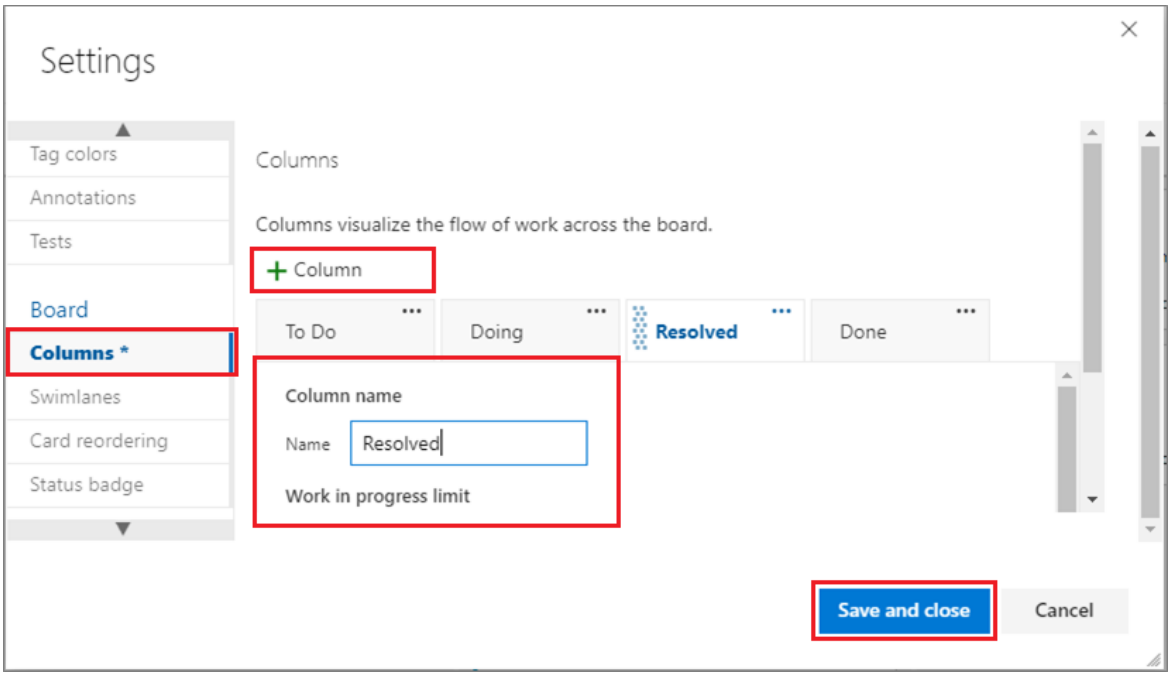
For this purpose, you will be adding '**Resolved**' column. If task is completed, move task to '**Resolved'** state and once reviewed then close.

Step 22:

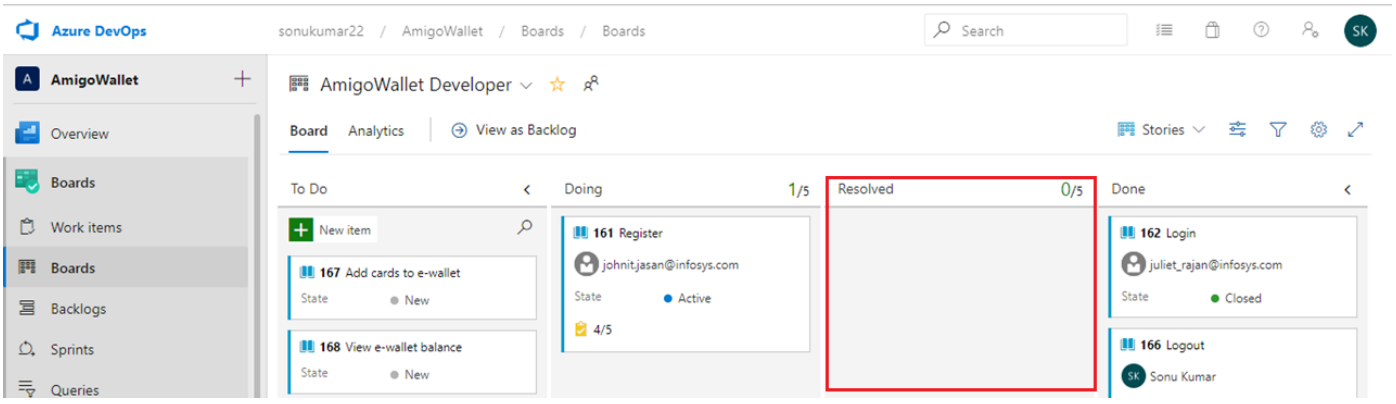
Repeat same step as 20.

Step 23:

Select Board > Columns. Select **Column** and give it name as '**Resolved**' and **Click Save and Close**.



Navigate to **Boards.**



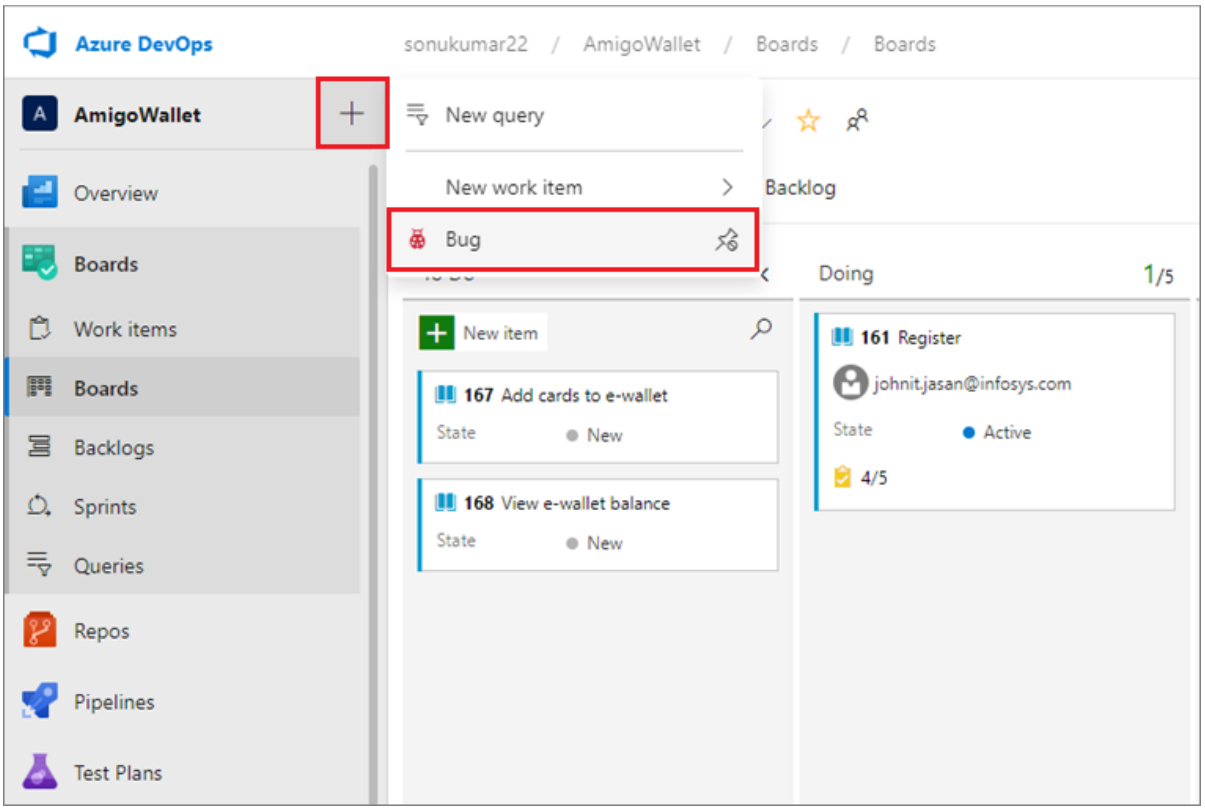
Observe **Resolved** column is added. Drag tasks that is completed to '**Resolved'** column or change the state to '**Resolved'**.

Adding Bugs as work item:

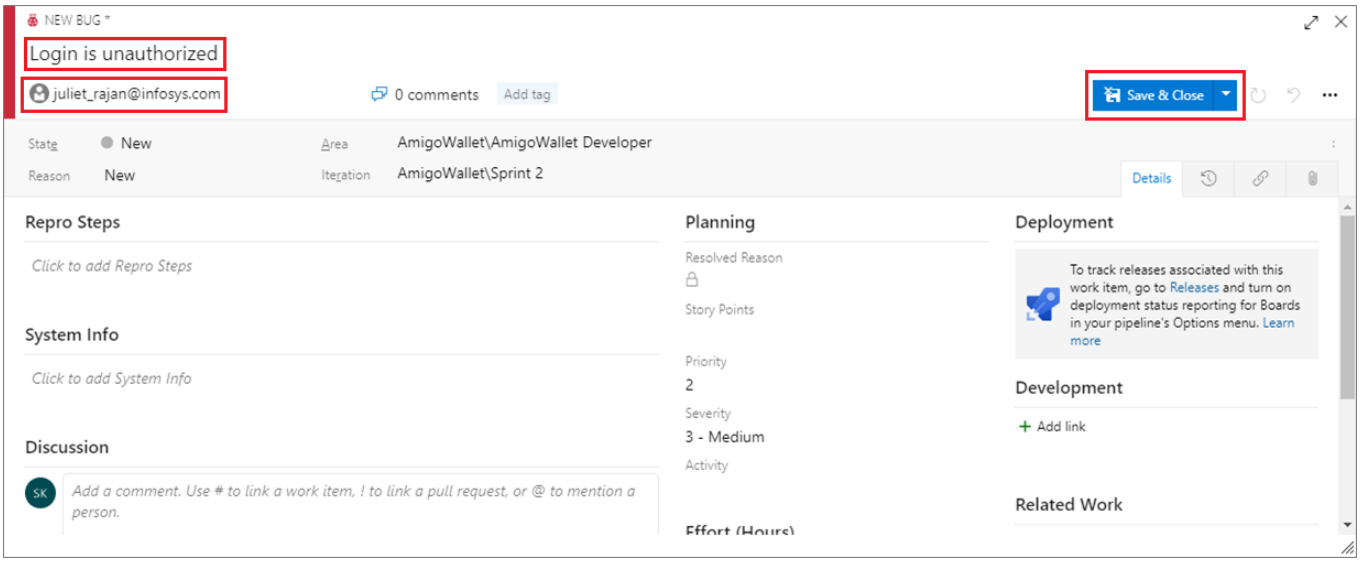
As your team identifies code defects or bugs, they can add them to the backlog and track them similar to requirements. Or, they can schedule them to be fixed within a sprint along with other tasks.

Step 24:

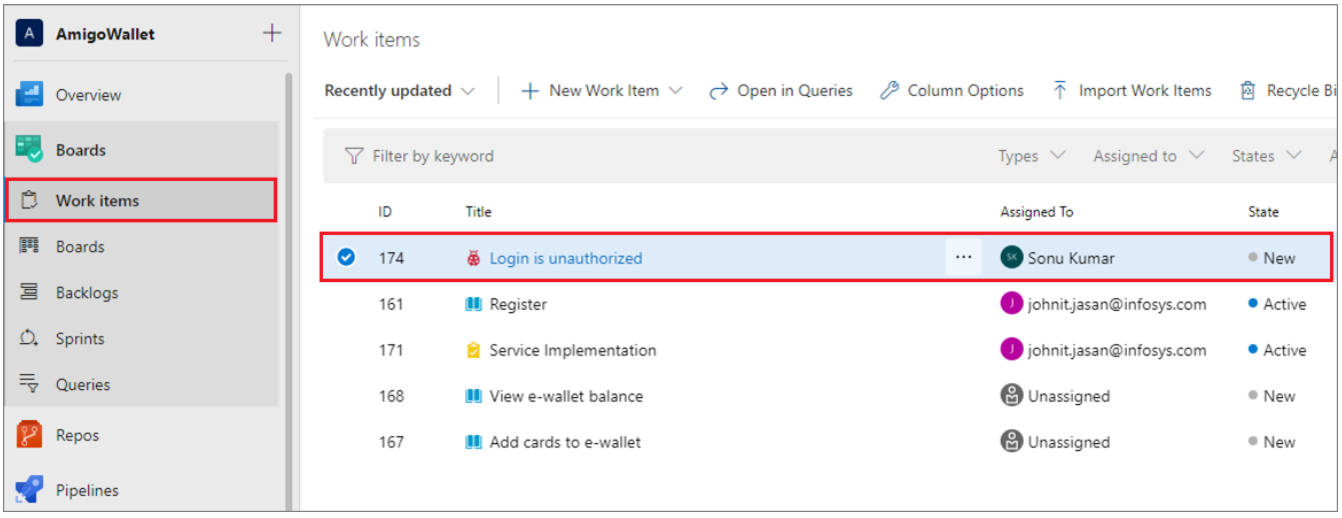
In order to add bug **click** + button right next to project name i.e. **AmigoWallet** then **select Bug.**



Give bug's **Name** as **Login is unauthorized** and also provide details for **Assigned to, Description** etc.



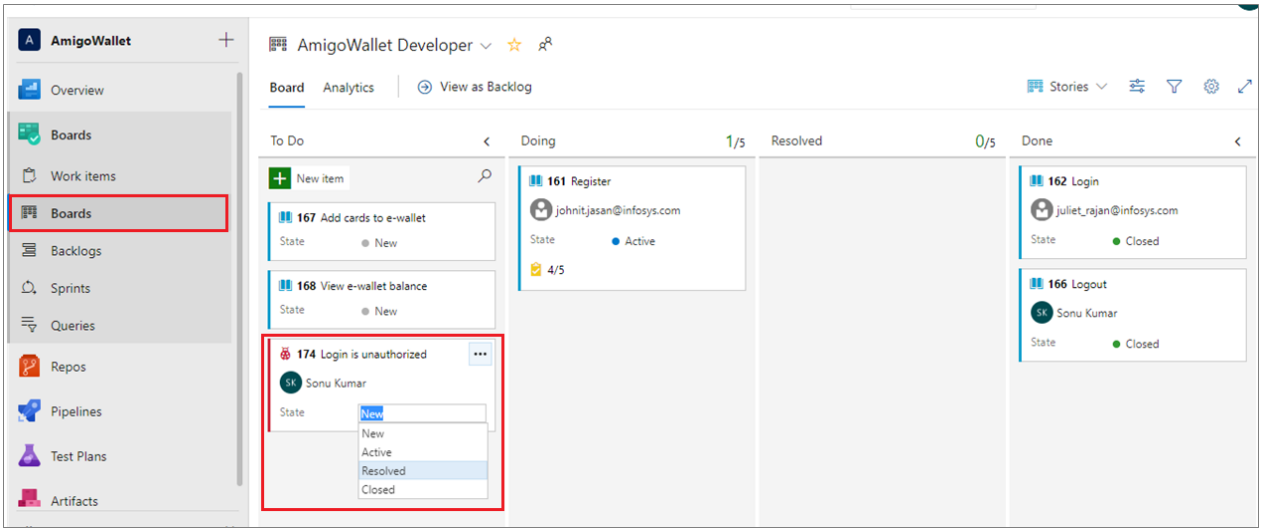
This bug will be added directly as work item but not inside any user story. Navigate to **Work Item**, all work items like User Story, Task, Feature, Bug will have listed here.



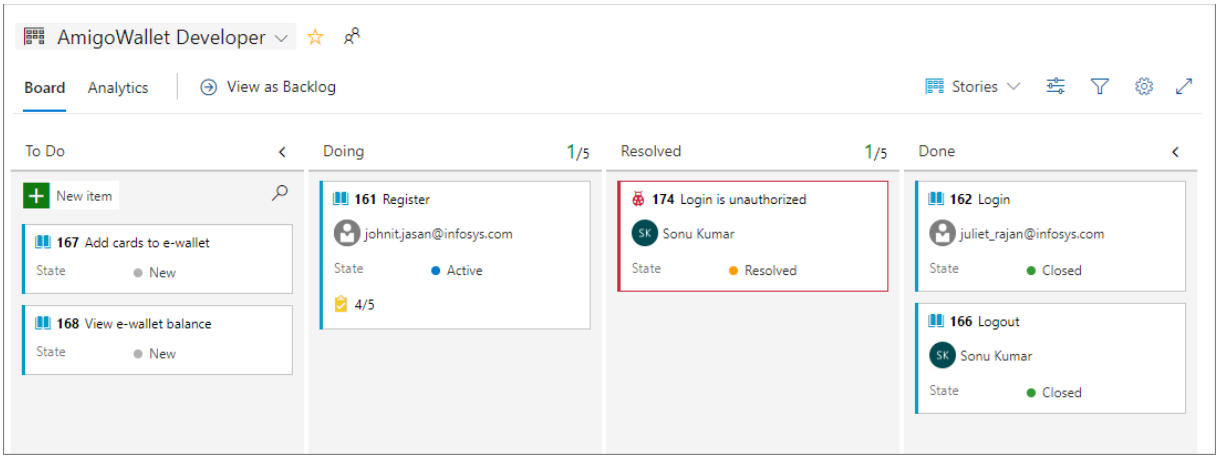
Observe that newly create bug '**Login is unauthorized**' is present here.

Step 25:

Navigate to **Boards**. You can select state of bug based on status.



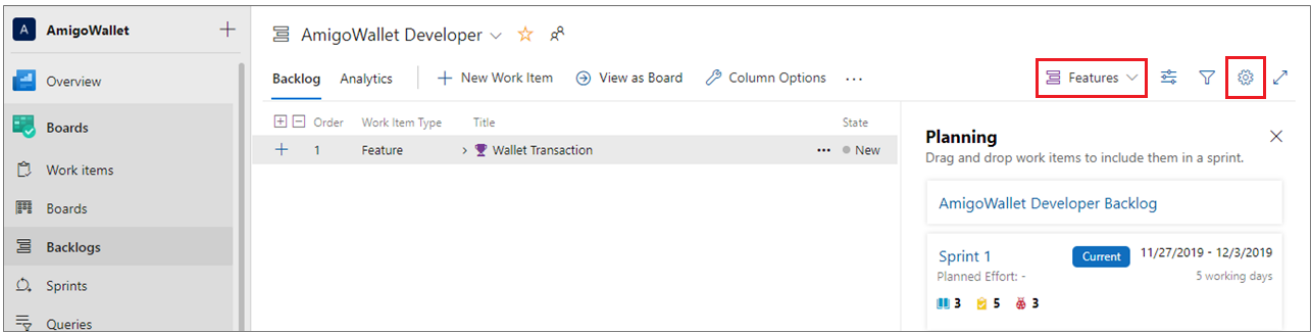
Select state of bug '**Login is unauthorized**' as **Resolved.**



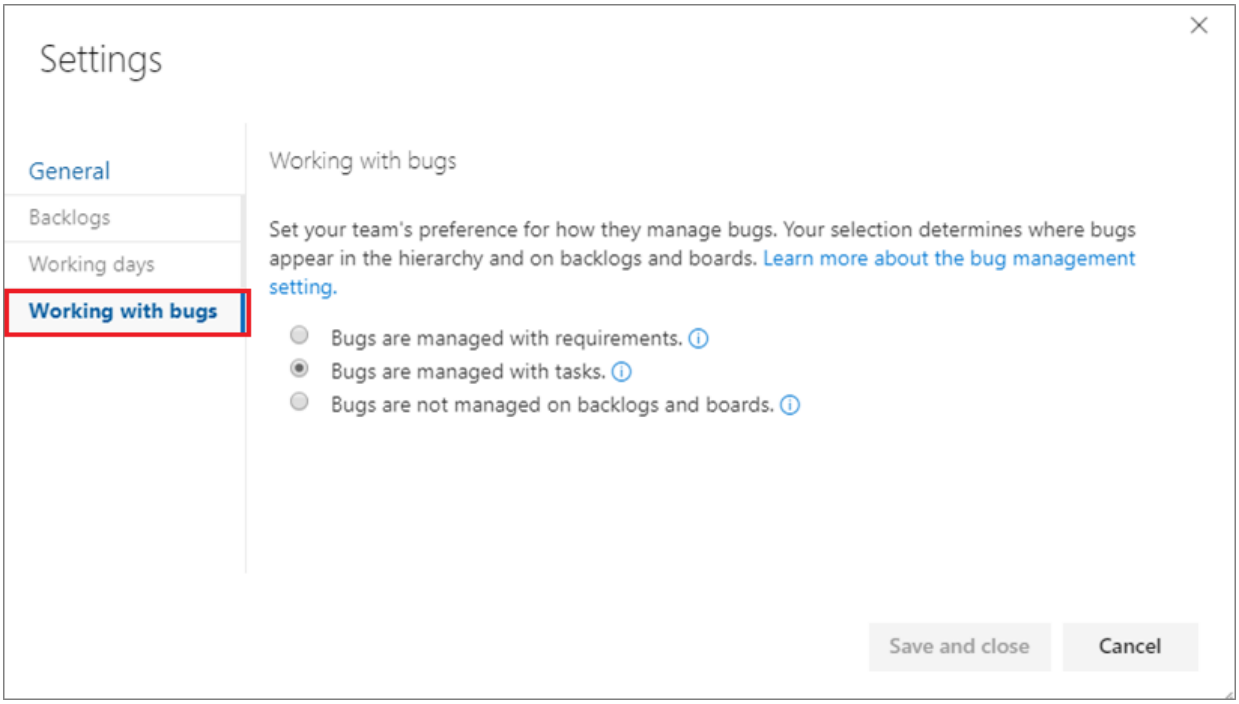
You can select how to manage bugs based on requirement.

Step 26:

Go to **Backlogs** or **Sprints. Select Settings**



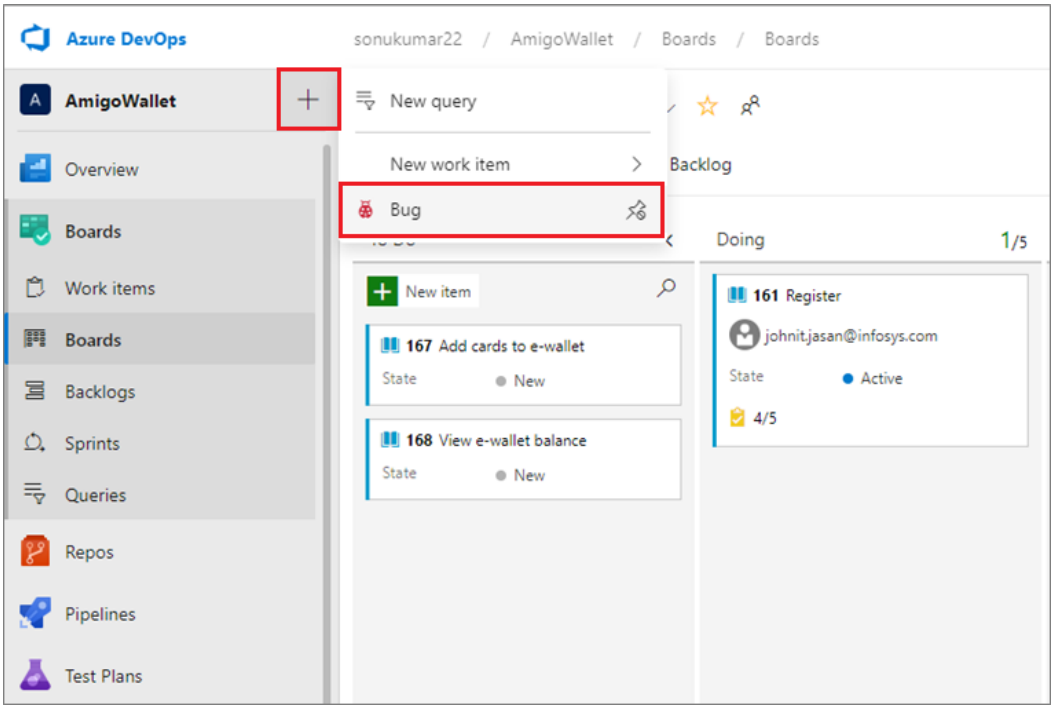
Select Working with bugs and choose '**Bugs are managed with tasks'** option. This enable bugs to be present on **board** and **backlog.**


Settings
General
Backlogs
Working days
Working with bugs
Working with bugs
Set your team's preference for how they manage bugs. Your selection determines where bugs appear in the hierarchy and on backlogs and boards. Learn more about the bug management setting.
Bugs are managed with requirements.
Bugs are managed with tasks.
Bugs are not managed on backlogs and boards.
Save and close
Cancel

Now let's try to add bugs specific to a user story. Here we will be adding bugs to '**Register**'.

Step 27:

In order to add bug **click** + button right next to project name i.e. **AmigoWallet** then **select Bug.**

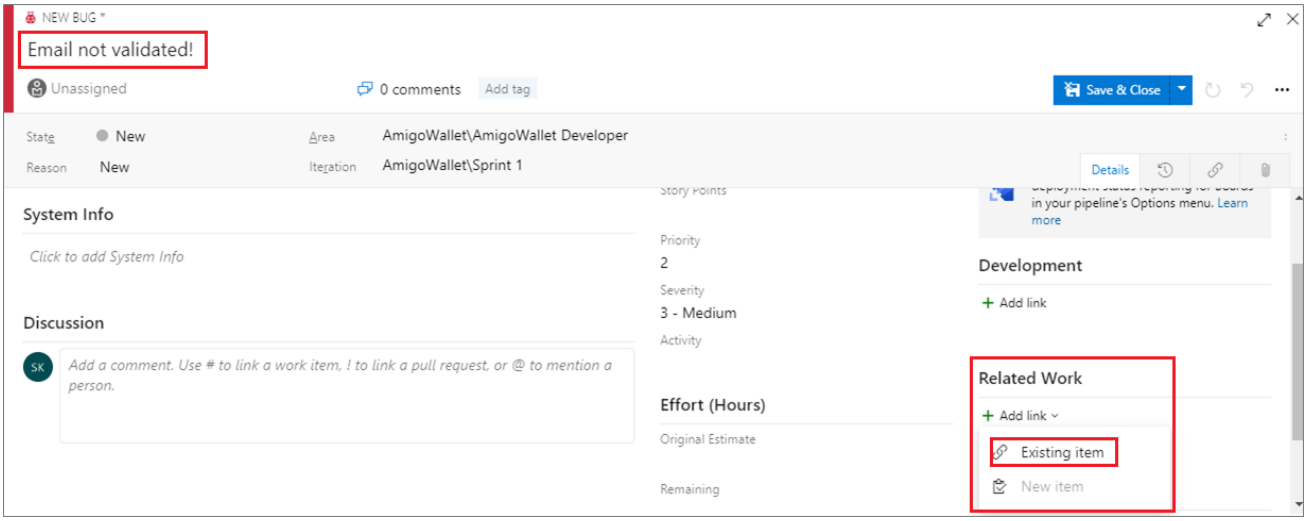


Give bug's **Name** as '**Email not validated!'** and also provide details for **Assigned to, Description** etc.

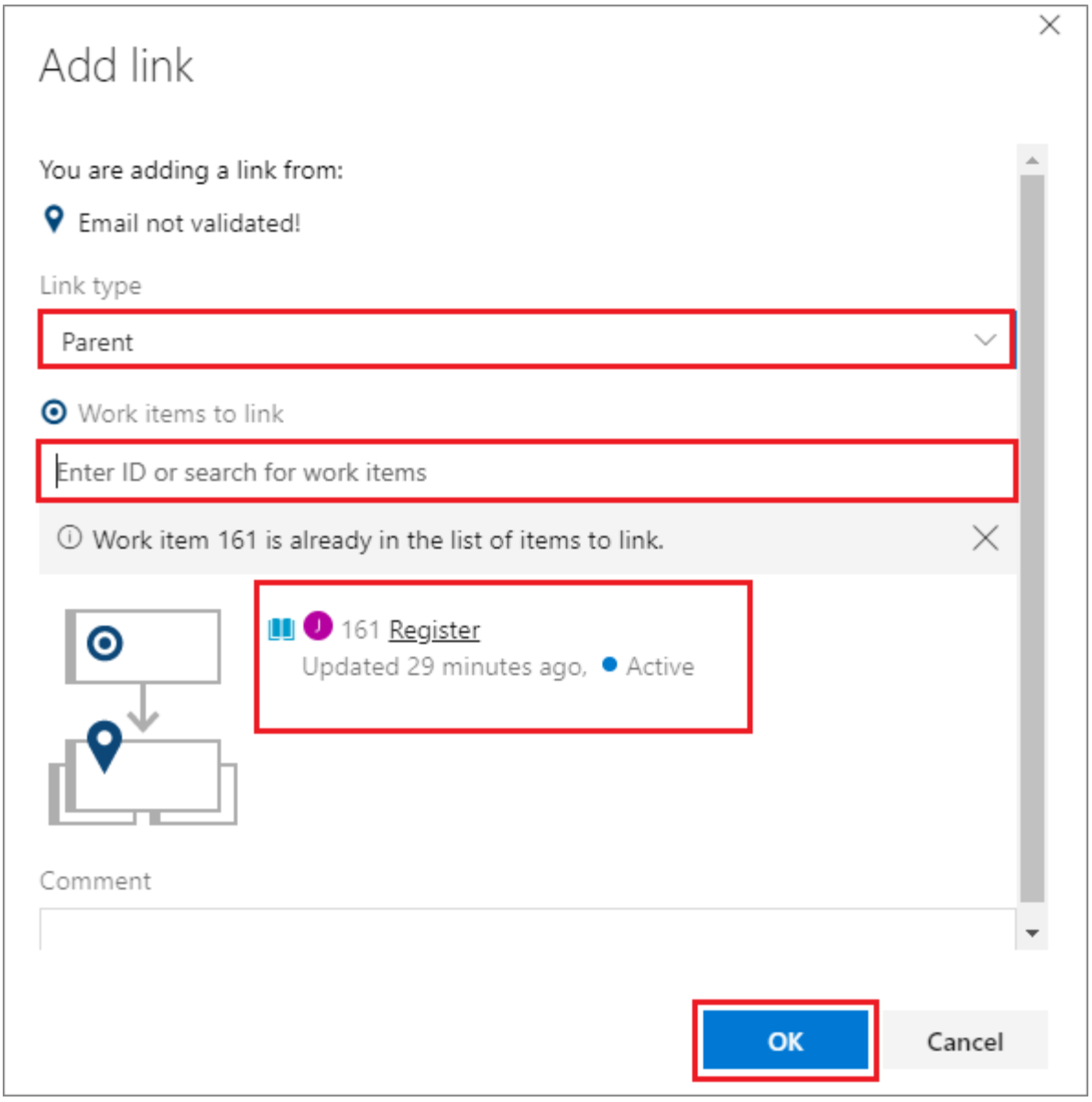
Here we want that bugs should be present inside **Register** user story i.e. bugs should be linked to **Register** as parent-child relation same as **Epic>Feature or Feature>User Story.**

For establishing relation between **Register** and '**Email not validated!'** bug

select Related Work > Add Link > Existing item.



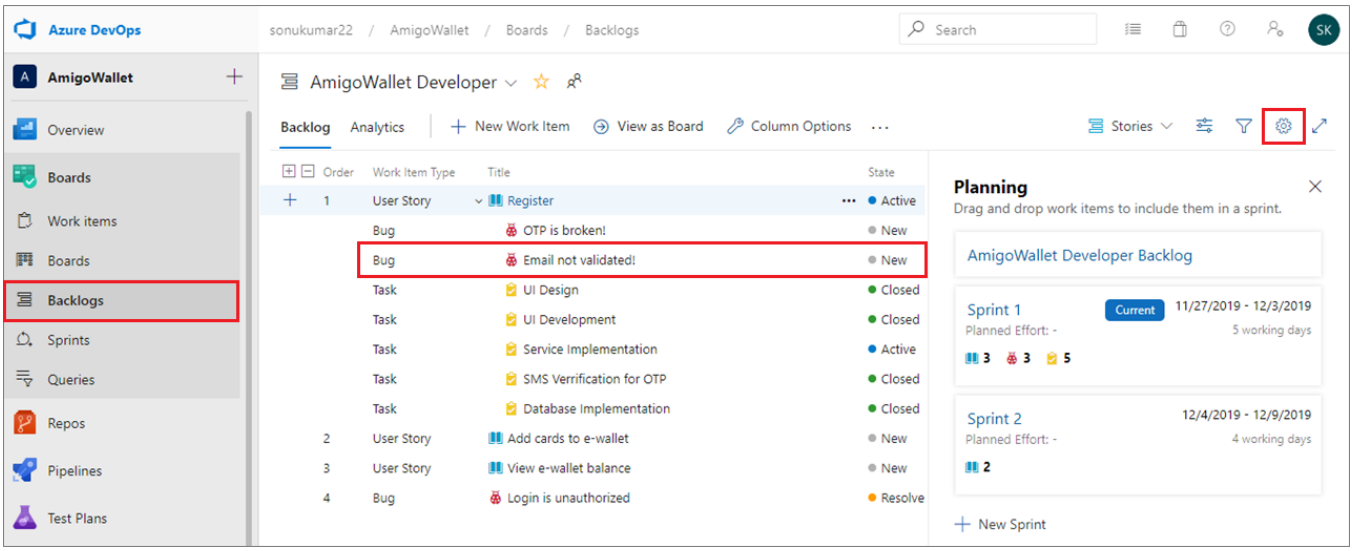
Select **Parent** from **Link Type** dropdown and '**Register** feature from **Work Items** and **click OK** then **Click Save and Close.**



Following same steps, add one more bug named '**OTP is broken!**' under **Register**.

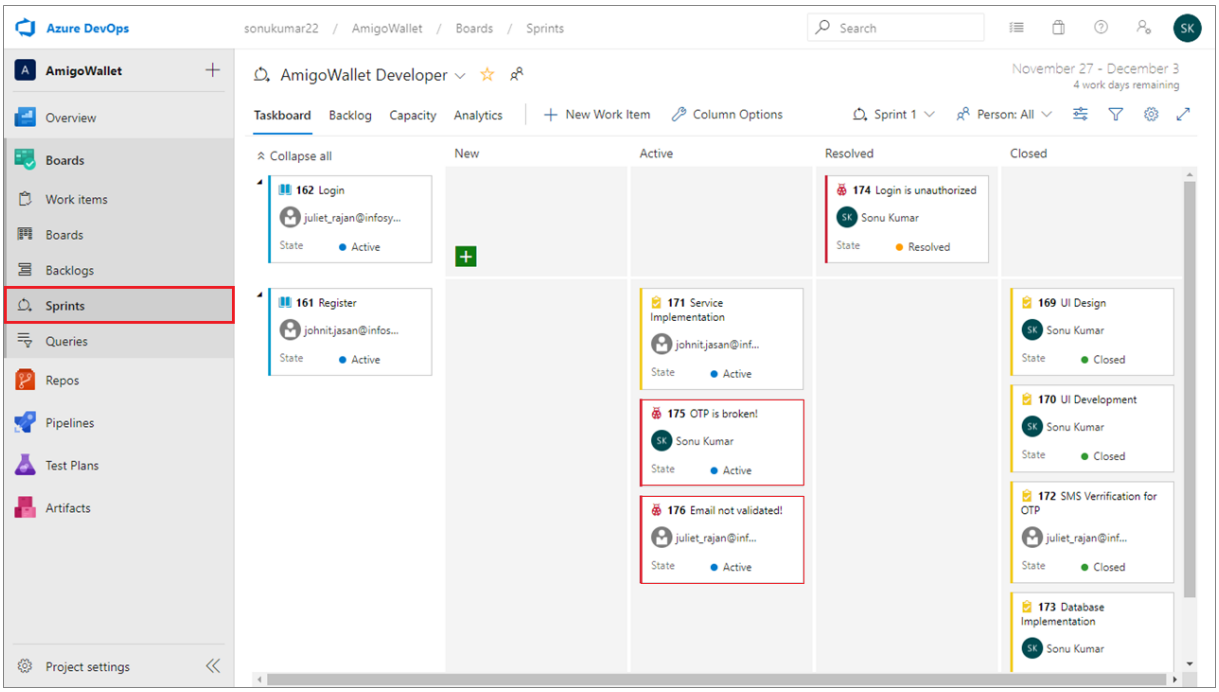
Step 28:

Go to **Backlogs. Expand Register.**



You can observe both bugs are present inside **Register.**

Navigate to **Sprints.** Change state of both bugs to **Active**.

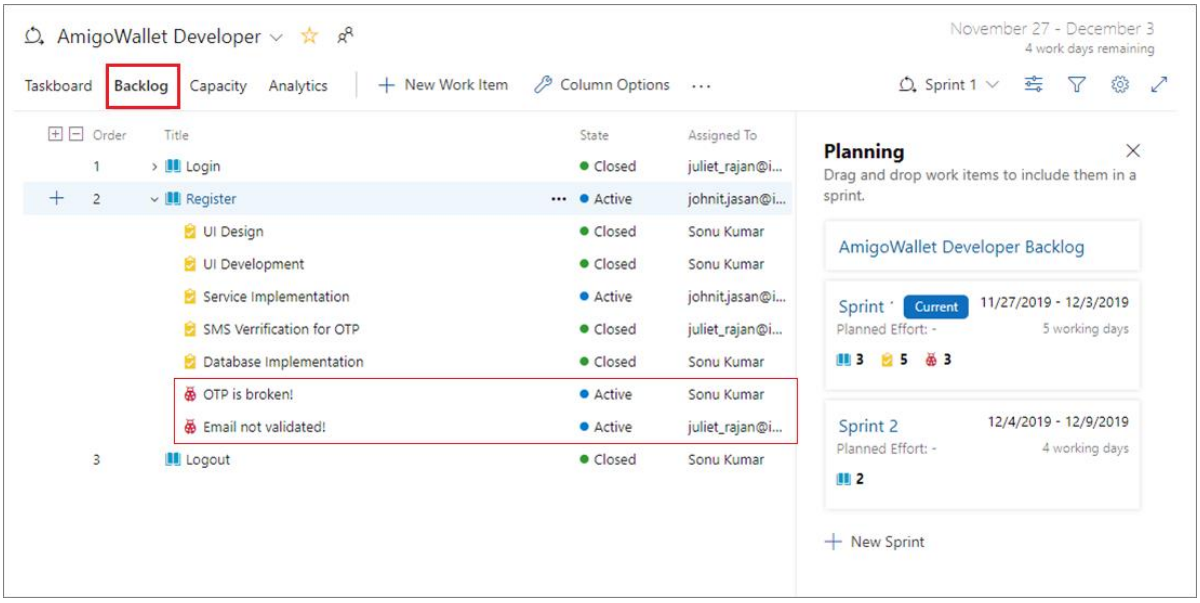


You can observe that both bugs are now moved to '**Active**' column.

If you want, you can make **Login is unauthorized** bug as part of **Login** user story in same way.

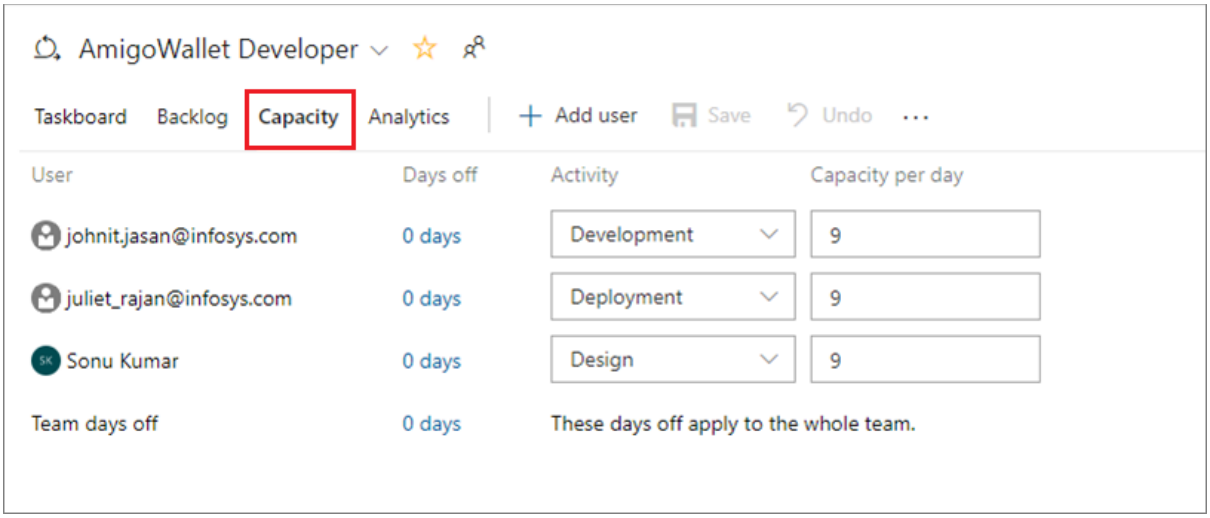
Step 29:

Navigate to **Backlog** tab present inside **Sprints.** User story related to particular sprint will be listed here.



Observe list of User story is only for sprint 1. If you want, see user stories for sprint 2 **select Sprint 2** from dropdown as mentioned in **step 20.**

Navigate to **Capacity** for capacity planning.

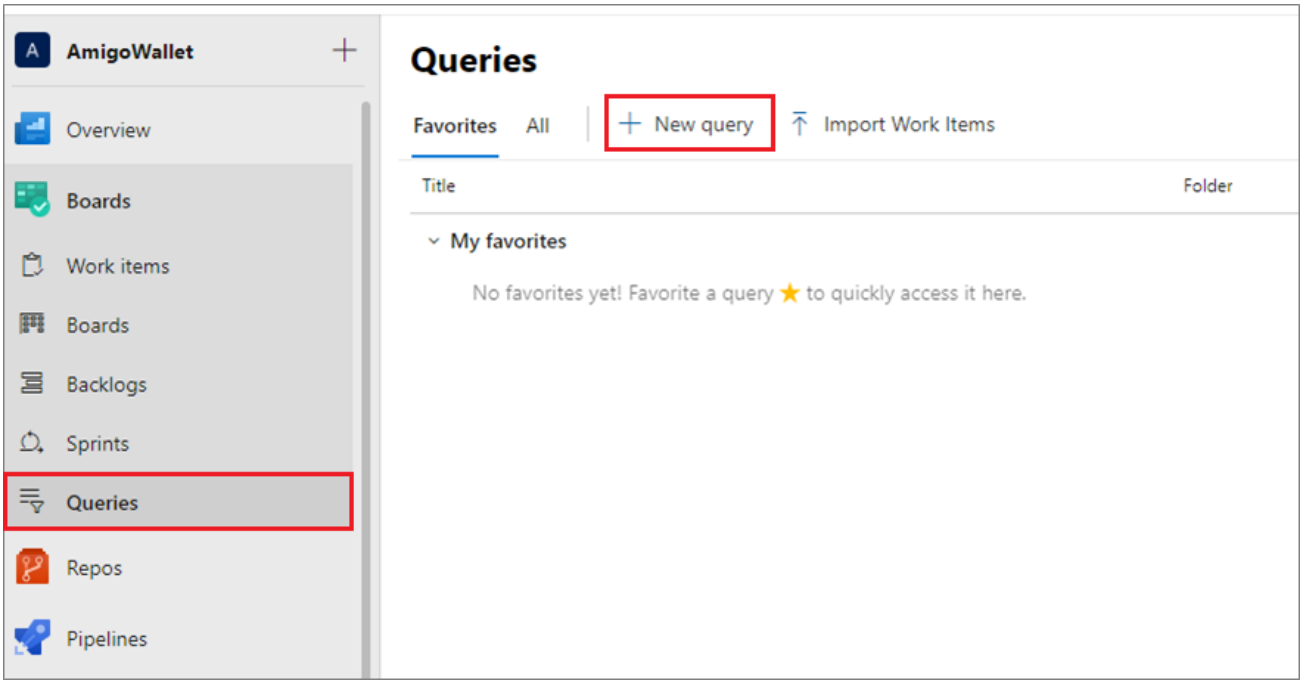


Queries

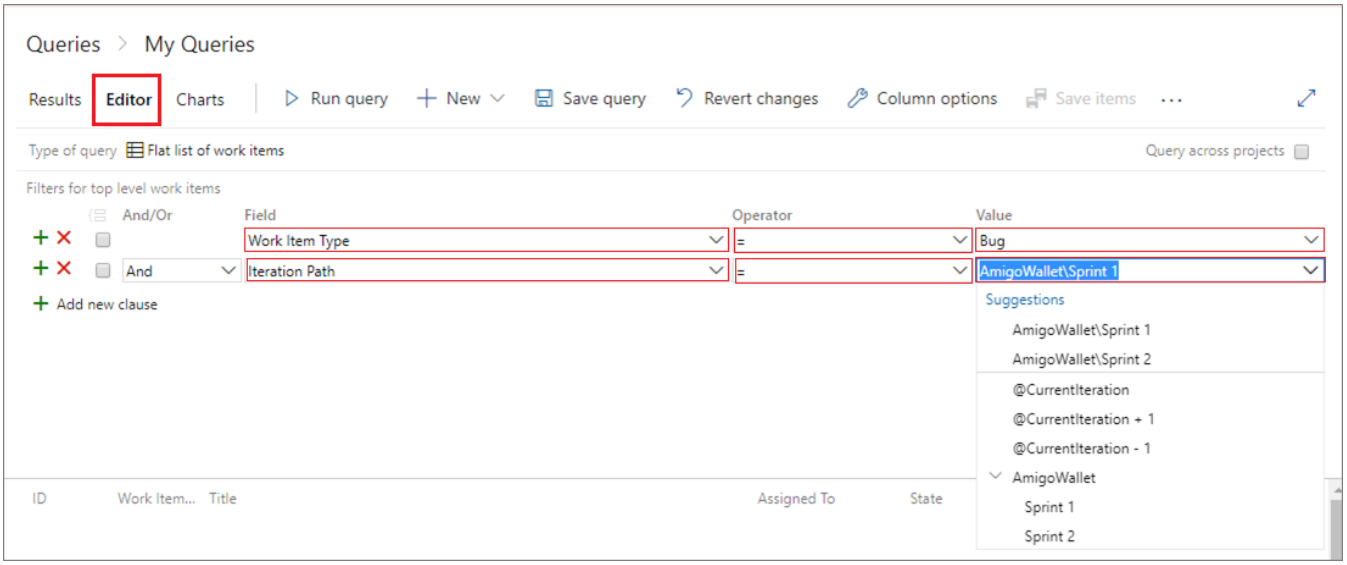
List bugs, user stories, or other work items based on field criteria you specify using queries. You can then review these lists with your team, triage work, or perform bulk work item updates.

Step 30:

Navigate to **Queries** section. **Select New Query**.



Go to **Editor** tab inside **Queries** section to write and execute queries.



Here you can select **Field Name, Operator** and **Value** for fields. You can even club two conditions by selecting second set of **Field Name, Operator** and **Value** for fields.

Here you are going to write a query to fetch all bugs present in Sprint1. So you need two conditions here:

Condition 1: Work Item Type = **Bugs.** Here **Work Item Type** is Field, = is operator and **Bugs** is value.

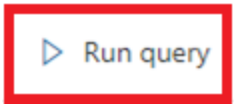
This query will give you all bugs of application

Condition 2: Iteration Path = **AmigoWallet/Sprint1.** Here **Iteration Path** is Field, = is operator and **AmigoWallet/Sprint1**is value.

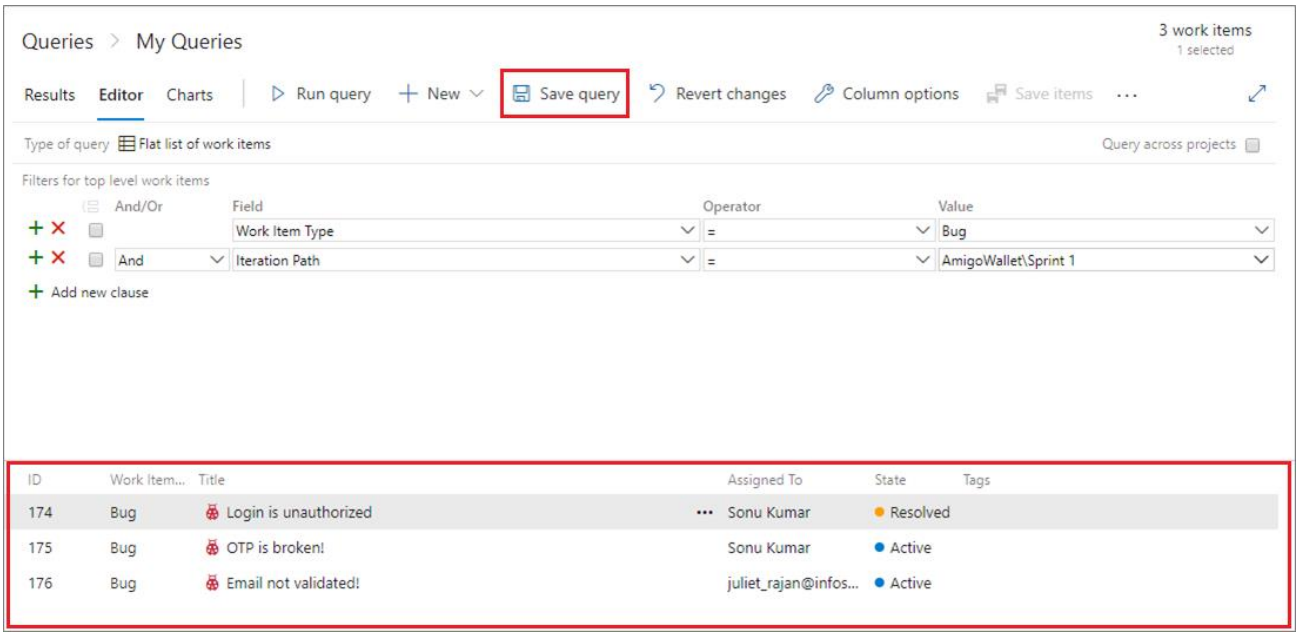
This query will give you all work items of sprint1.

If you club both condition you will get bugs for sprint1.

Click Run Query



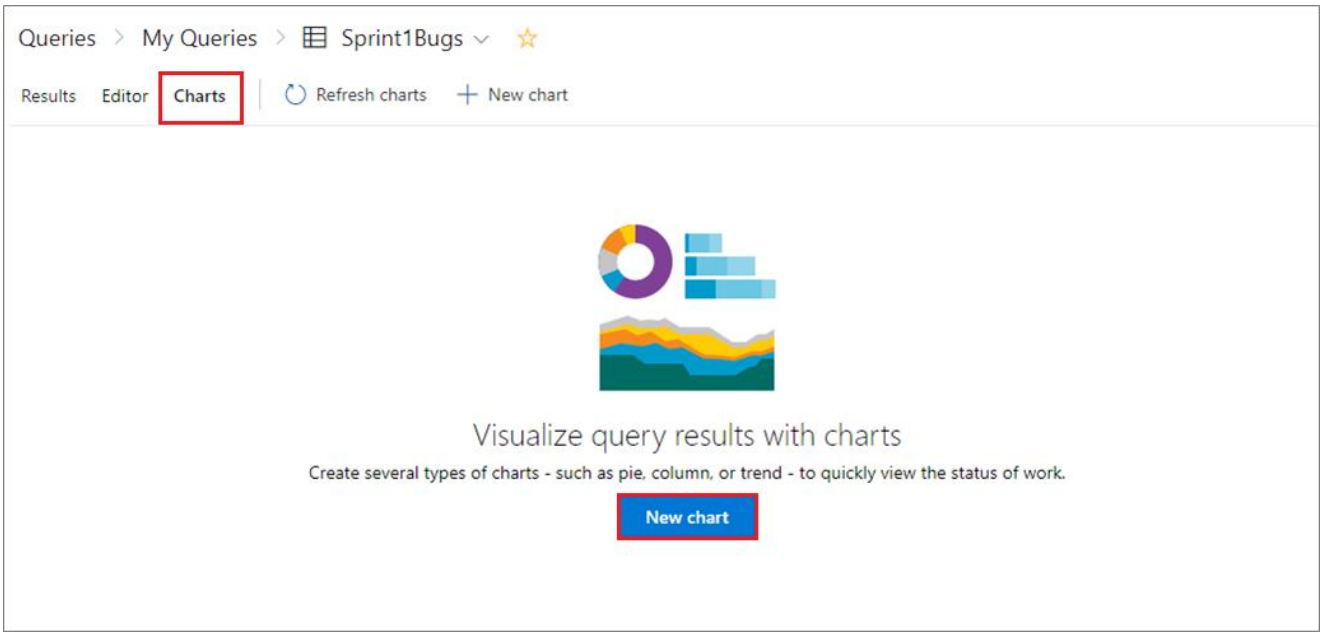
You can observe result of query at bottom. Save query to folder using **Save Query** button so that you can use query later. Give it name as **Sprint1Bugs**



You can create charts for saved queries for better experience or to include that on Dashboard of Project.

Step 31:

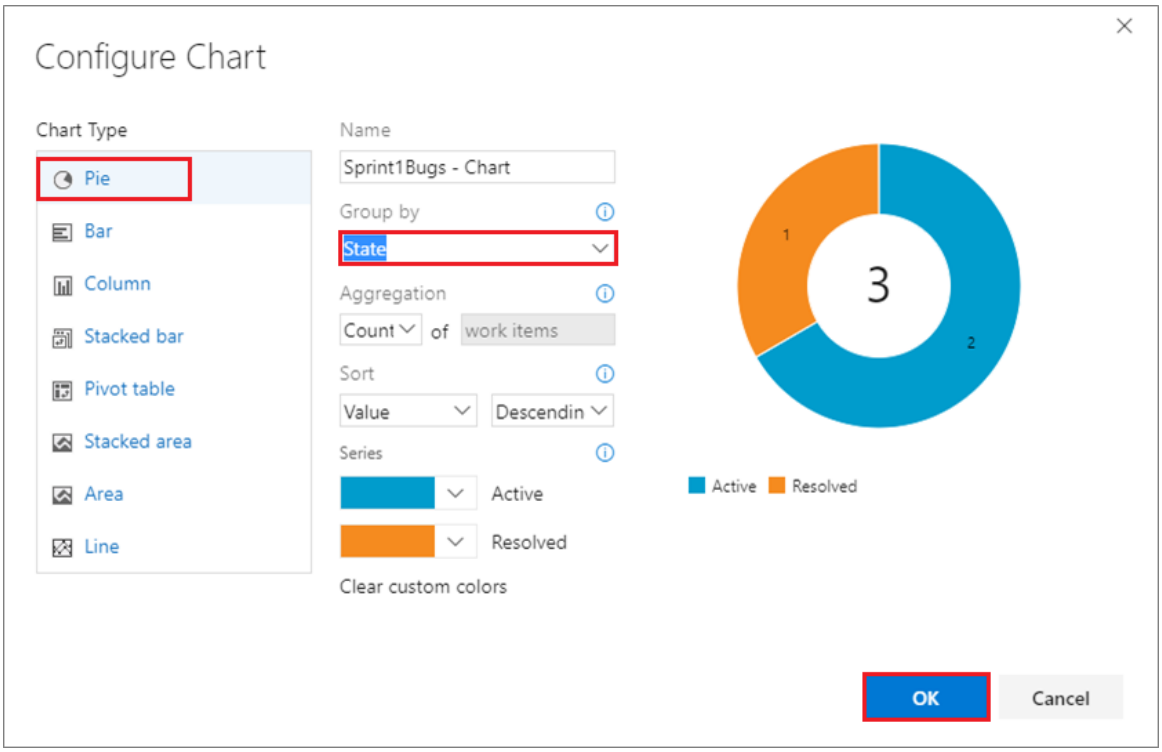
Go to Chart inside **Queries** section. Click on **New Chart.**



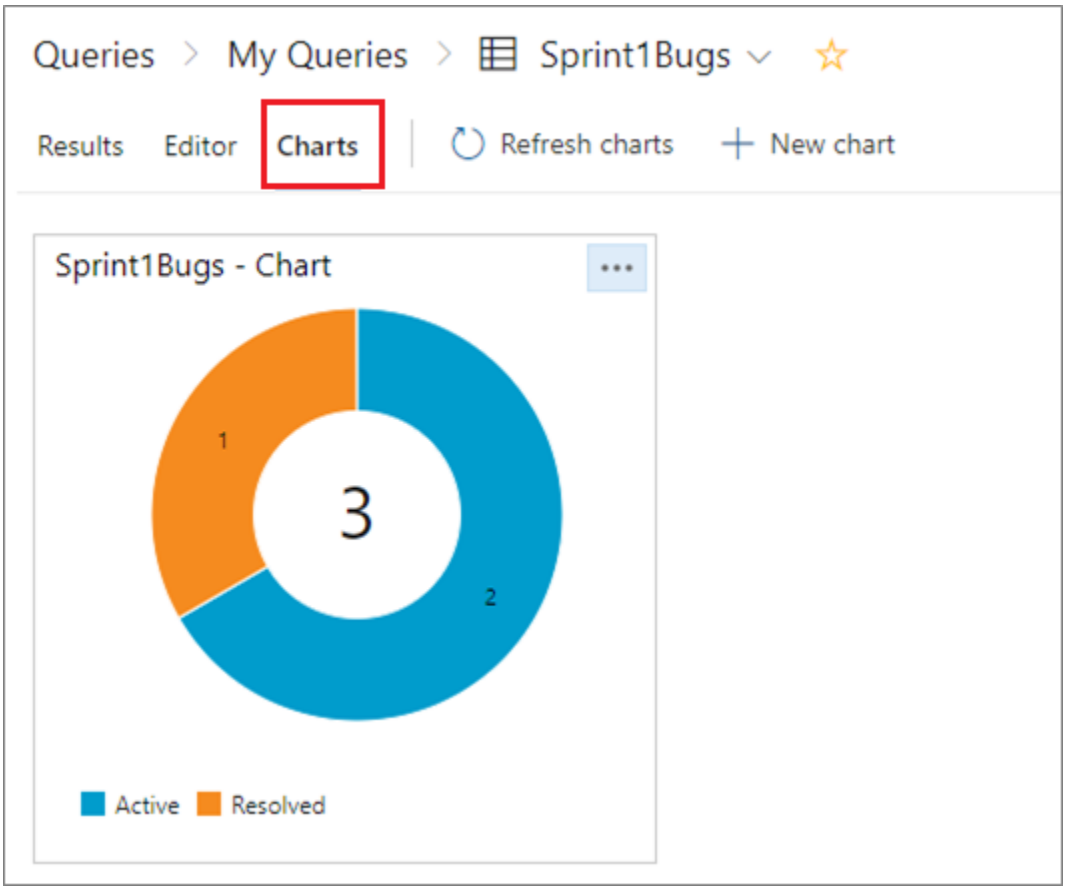
You can type of charts depending on scenarios.

Select **pie** chart and select **Group by** as **state** and Name as **Sprint1Bugs.**

This will give you a pie chart for all bugs grouped by state of bug (New, Active, Resolved, Closed).

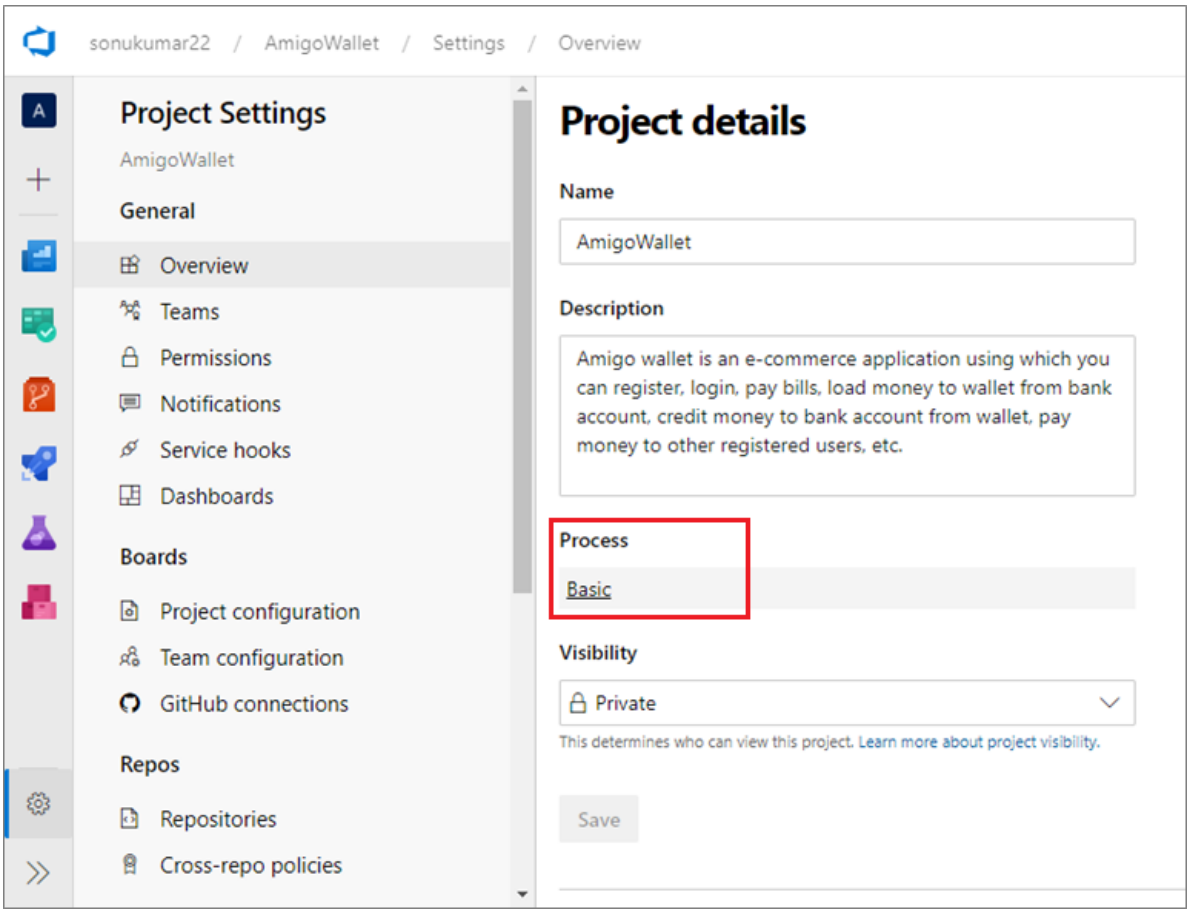


Click OK. Chart newly created chart will be present inside **Chart** tab.



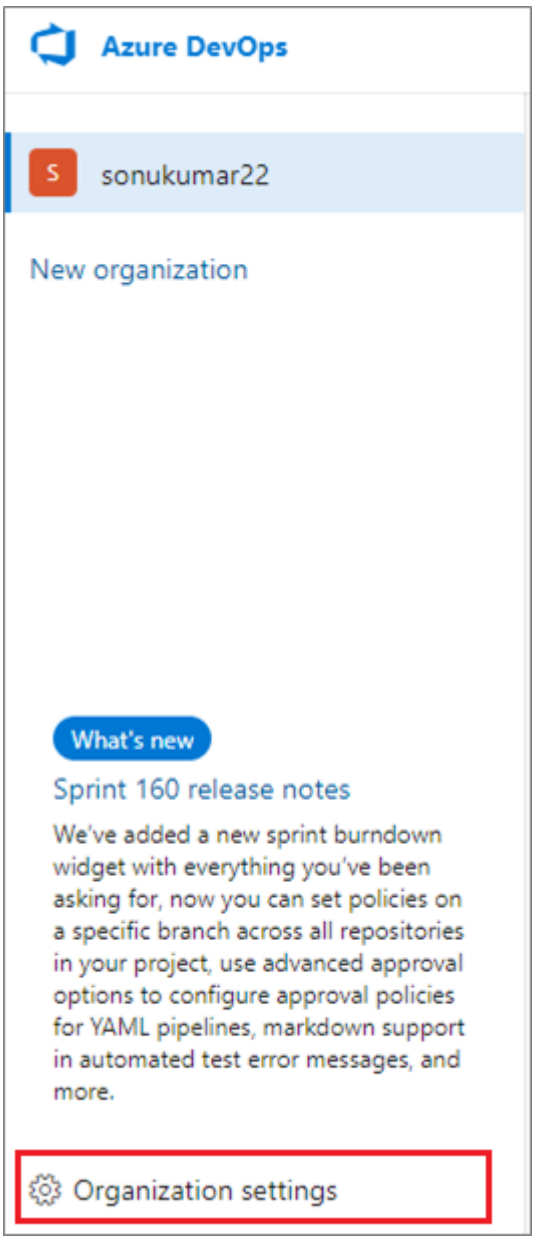
Switching Project process from Basic to Agile.

You can observe here process of project is **Basic**.

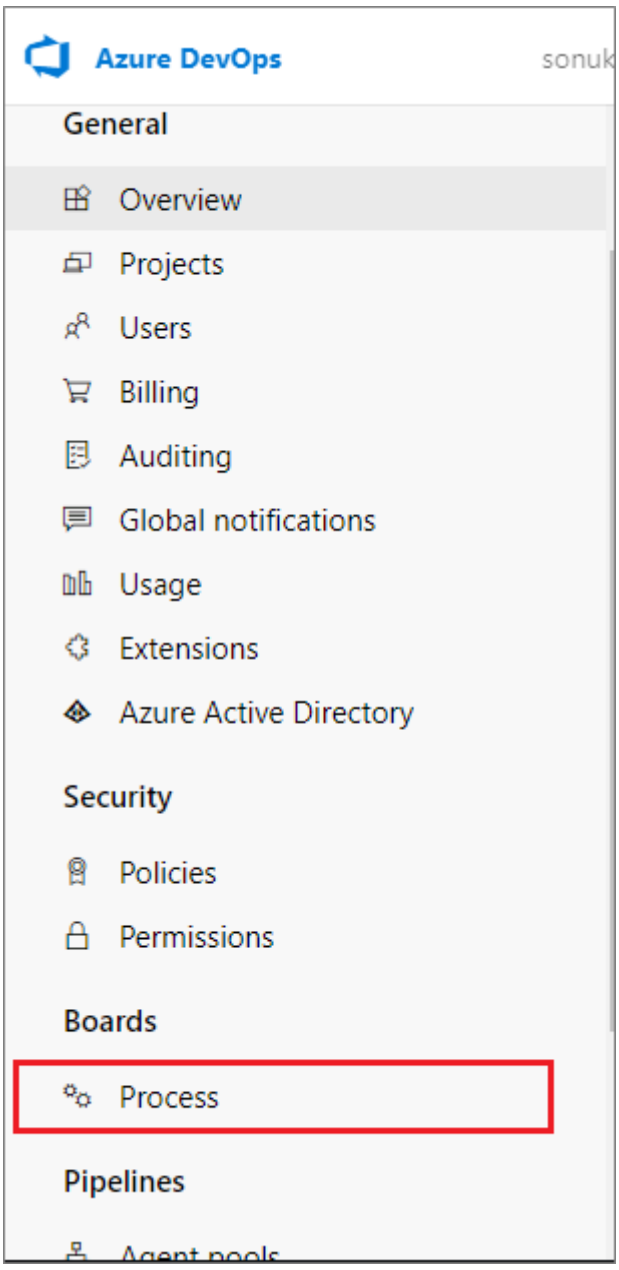


Step 1:

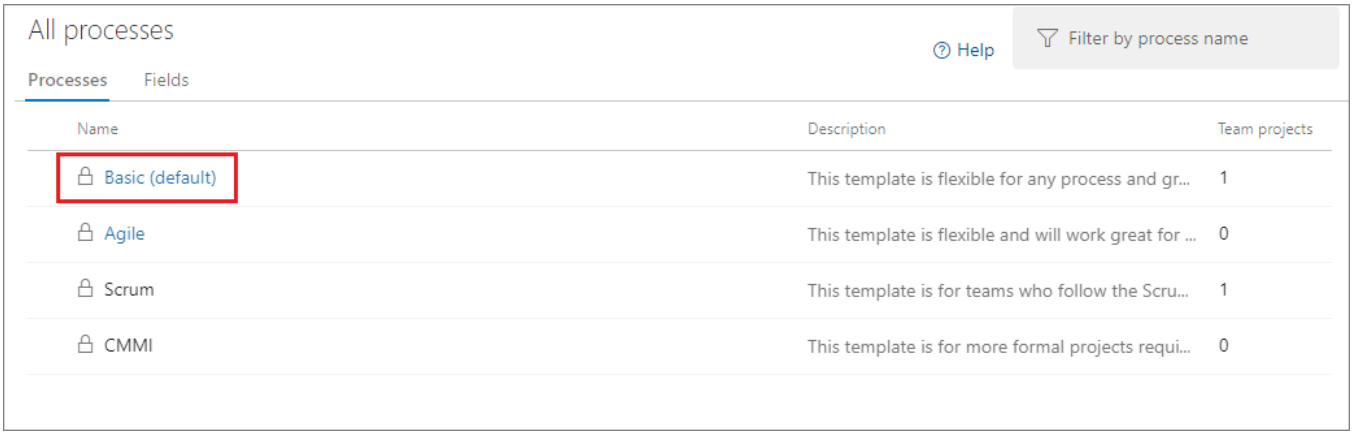
In order to change go to **Organization settings.**


Azure DevOps
S
sonukumar22
New organization
What's new
Sprint 160 release notes
We've added a new sprint burndown widget with everything you've been asking for, now you can set policies on a specific branch across all repositories in your project, use advanced approval options to configure approval policies for YAML pipelines, markdown support in automated test error messages, and more.
Organization settings

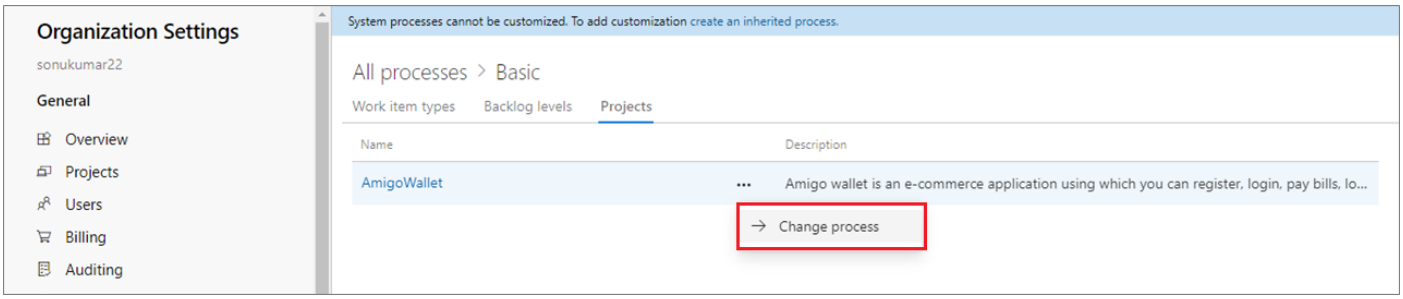
Select **Boards>Process**



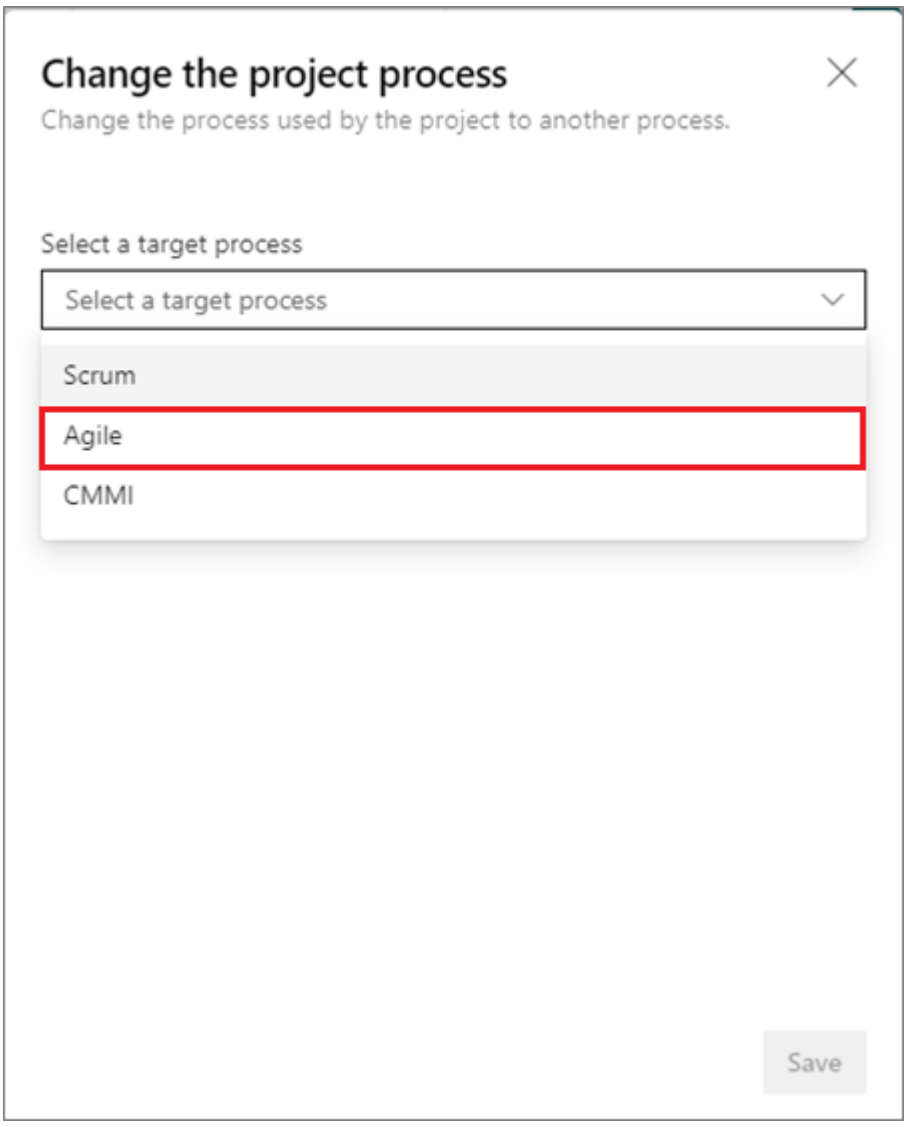
Select **Basic(Default).**



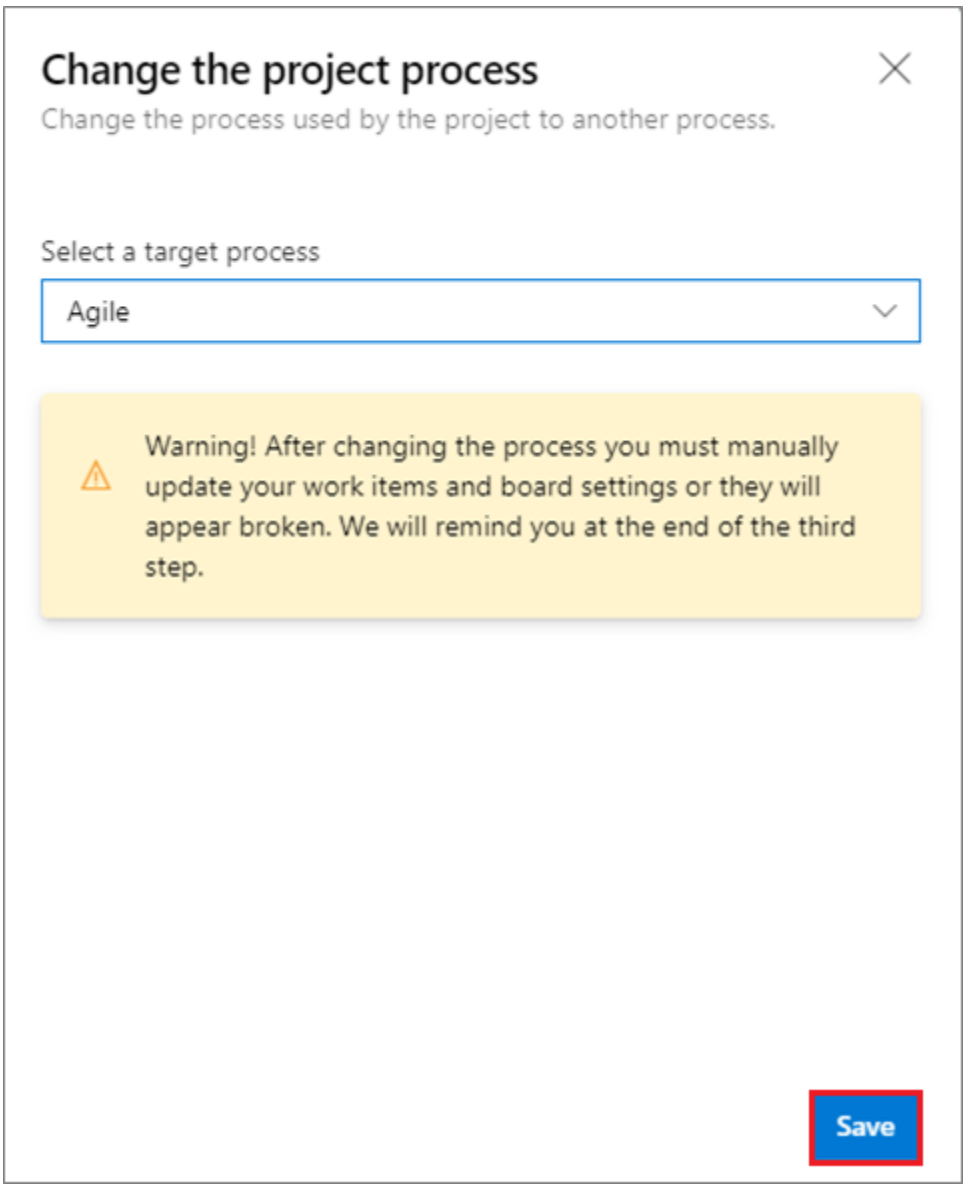
Go to Projects tab and choose **Change process** by **clicking** on ••• actions icon**.**



Select target process as Agile from dropdown.

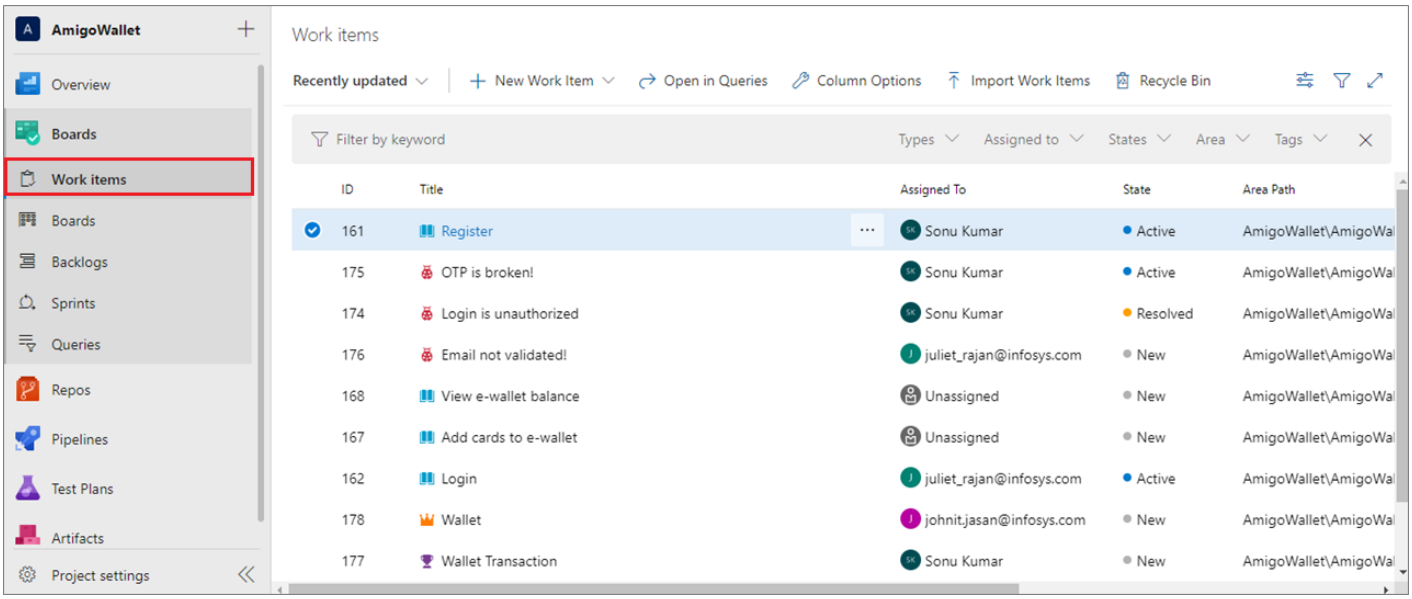


This will give you warning if you have already implemented boards. **Click on Save.**


Change the project process
Change the process used by the project to another process.
Select a target process
Agile
Warning! After changing the process you must manually update your work items and board settings or they will appear broken. We will remind you at the end of the third step.
Save

This will change the process of project from **Basic(Default)** to **Agile**.

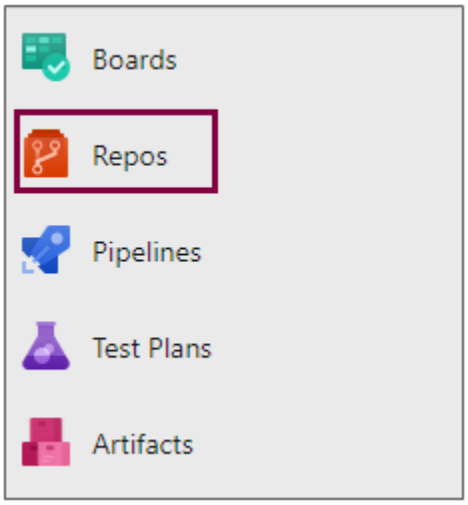
Every work item that you have added is visible in **Work Item** Section irrespective of any hierarchy.



Azure Repos

Step 1:

Navigate to dev.azure.com. Open the Repos of the project you created.



Step 2:

You will see the below commands. These two commands are used to push the on-premise "Amigo Wallet" application to the Repos.

Step 3:

Open the Node.js command prompt. Navigate to the local git repository directory path of your application "Amigo Wallet"

and execute the commands mentioned in the previous step.

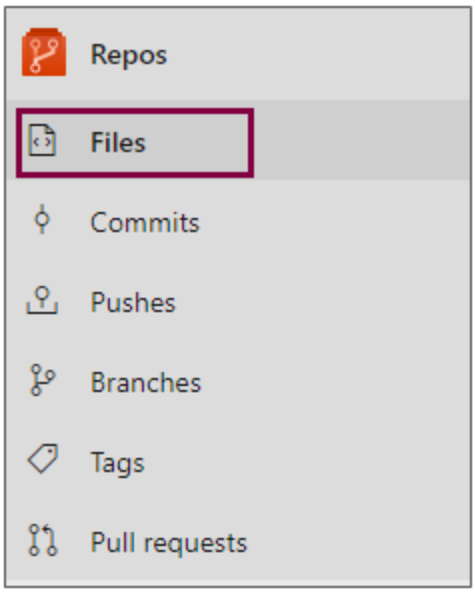
Administrator: Node.js command prompt
C:\AmigoWallet>git remote add origin https://julietrajan0767@dev.azure.com/julietrajan0767/AmigoWallet/_git/AmigoWallet
C:\AmigoWallet>git push -u origin --all

You will see the below on successful execution of the command.

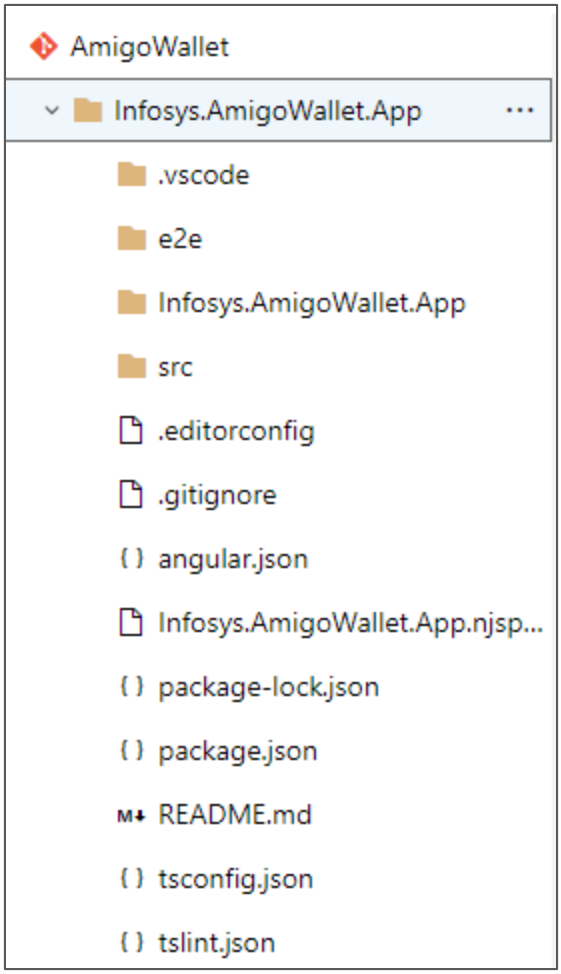
```
C:\AmigoWallet>git push -u origin1 --all
Enumerating objects: 198, done.
Counting objects: 100% (198/198), done.
Delta compression using up to 2 threads
Compressing objects: 100% (189/189), done.
Writing objects: 100% (198/198), 677.83 KiB | 4.26 MiB/s, done.
Total 198 (delta 32), reused 0 (delta 0)
remote: Analyzing objects... (198/198) (652 ms)
remote: Storing packfile... done (74 ms)
remote: Storing index... done (39 ms)
To https://dev.azure.com/julietrajan0767/AmigoWallet/_git/AmigoWallet
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin1'.
```

Step 4:

To verify that the application is pushed to the Repos, open the "File".



You will see the below.



Thus you have seen how to push the artifacts into the Repos.

If each developer wants to have a working copy created for him so that the master copy is not modified, the Repos can always be cloned. In order to clone the repo, you will do the following steps.

Step 1:

Navigate to the folder where you want this application to be cloned and execute the clone command as given below.

```
C:\AmigoWallet\Workingfolder>git clone https://julietrajan0767@dev.azure.com/julietrajan0767/AmigoWallet/_git/AmigoWalle
t
Cloning into 'AmigoWallet'...
remote: Azure Repos
remote: Found 198 objects to send. (43 ms)
Receiving objects: 100% (198/198), 677.83 KiB | 1.08 MiB/s, done.
Resolving deltas: 100% (32/32), done.
```

Iterations

Step 2:

You will see the entire application is cloned on your local drive.

If the developer is asked to include a production build in the package.json, he would want to test the changes before he modifies the master copy. He can create a "development" branch, work on it, test it and finally merge the changes in "development" branch with the "master" branch. You will see how to create branch and merge in the following.

Step 1:

To create a new branch, execute the below command

git branch development

Then push the changes into the "development" branch using the below command.

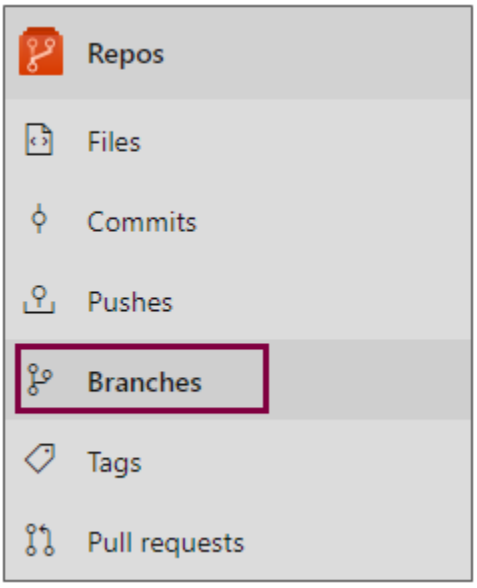
git push -u origin --all

```
C:\AmigoWallet\Workingfolder>git branch development

C:\AmigoWallet\Workingfolder>git commit -m "Modified the package.json file"
[master 1a9915b] Modified the package.json file
 1 file changed, 1 insertion(+)
 create mode 160000 Workingfolder/AmigoWallet

C:\AmigoWallet\Workingfolder>git push -u origin1 --all
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 355 bytes | 177.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: Analyzing objects... (3/3) (6 ms)
remote: Storing packfile... done (107 ms)
remote: Storing index... done (25 ms)
To https://dev.azure.com/julietrajan0767/AmigoWallet/_git/AmigoWallet
   7dacf48..1a9915b  master -> master
 * [new branch]      development -> development
Branch 'master' set up to track remote branch 'master' from 'origin1'.
Branch 'development' set up to track remote branch 'development' from 'origin
```

Thus a “development” branch is created and the changes are pushed into the development branch.

Step 2:

In order to merge the changes with the master branch, click on “Branches”.

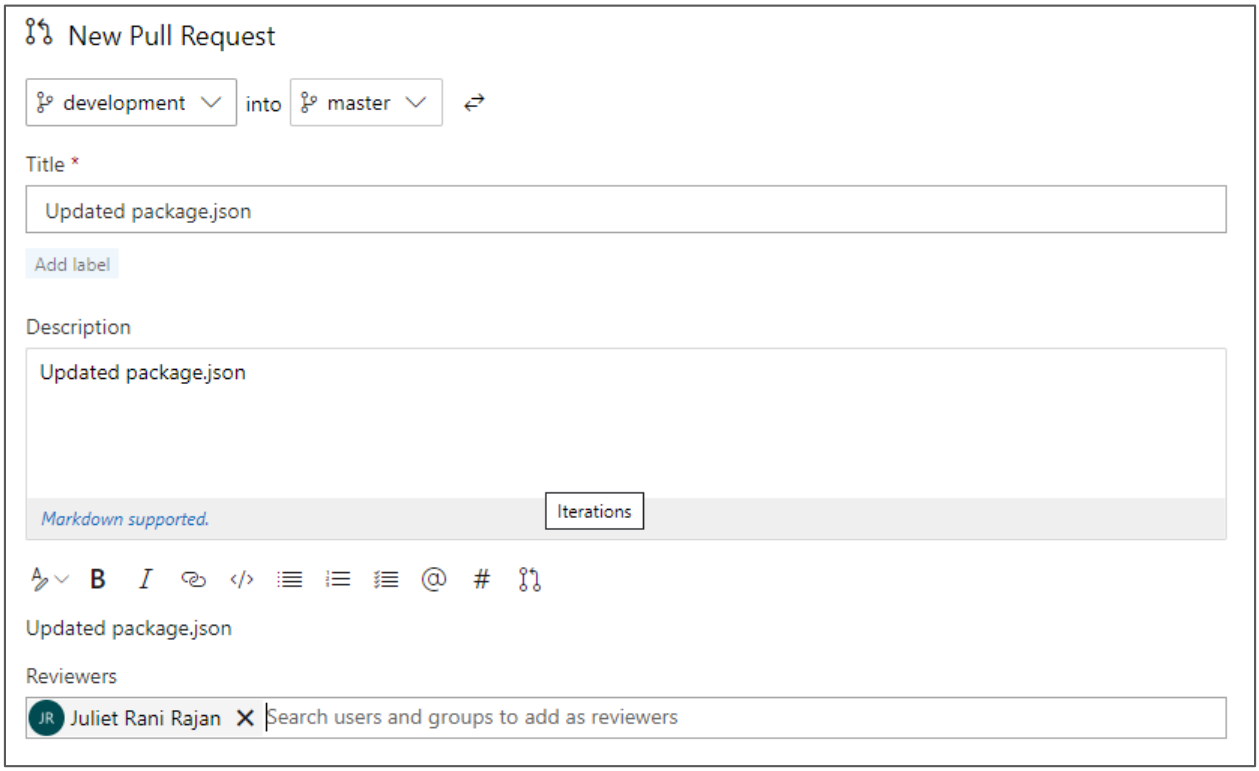


You will see the “development” branch you pushed before sometime.

Step 3:

Click on “Pull Request” of the “development” branch. Before merging the changes, you will create a “Pull Request”. Add appropriate titles and descriptions.

Also, specify the “Reviewers”.

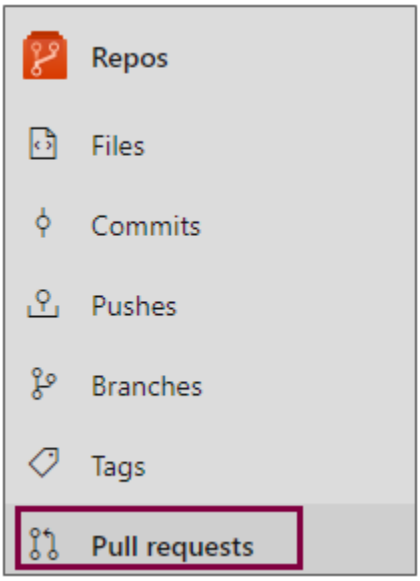


The Pull Request is now in the Reviewer’s inbox.

Note: In this case, Pull Request creator and the reviewer are the same.

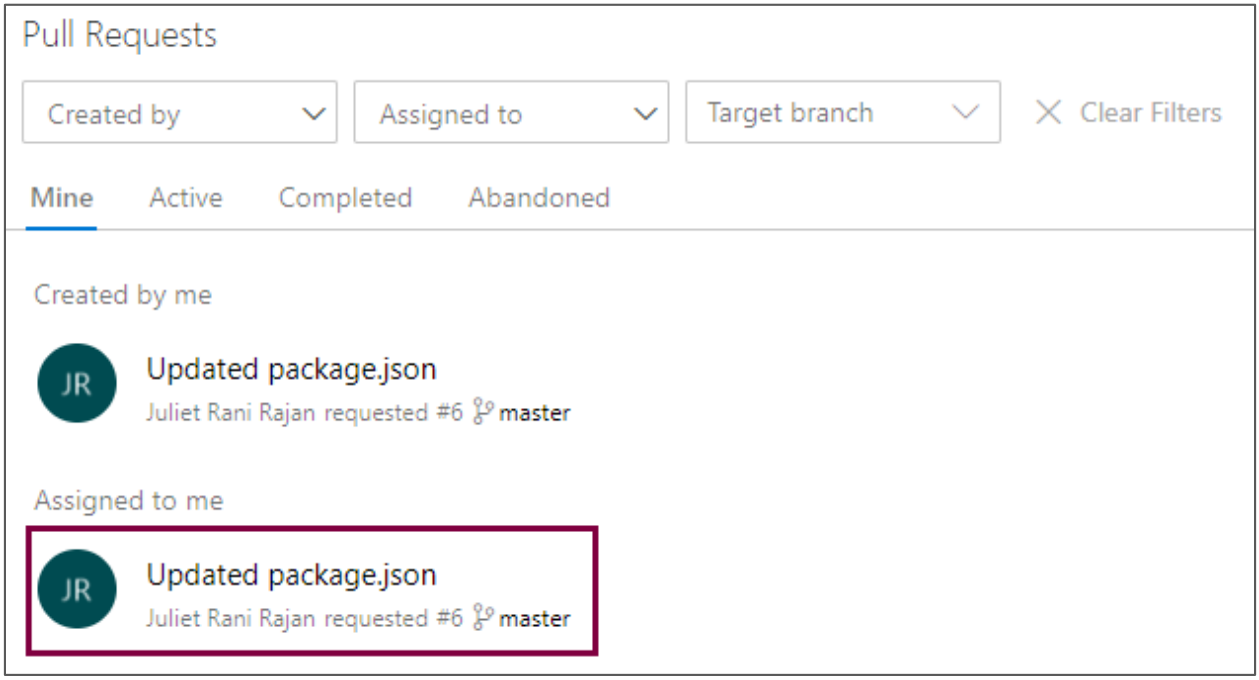
Step 4:

In order to view the “Pull Request”, click on “Pull Request”.

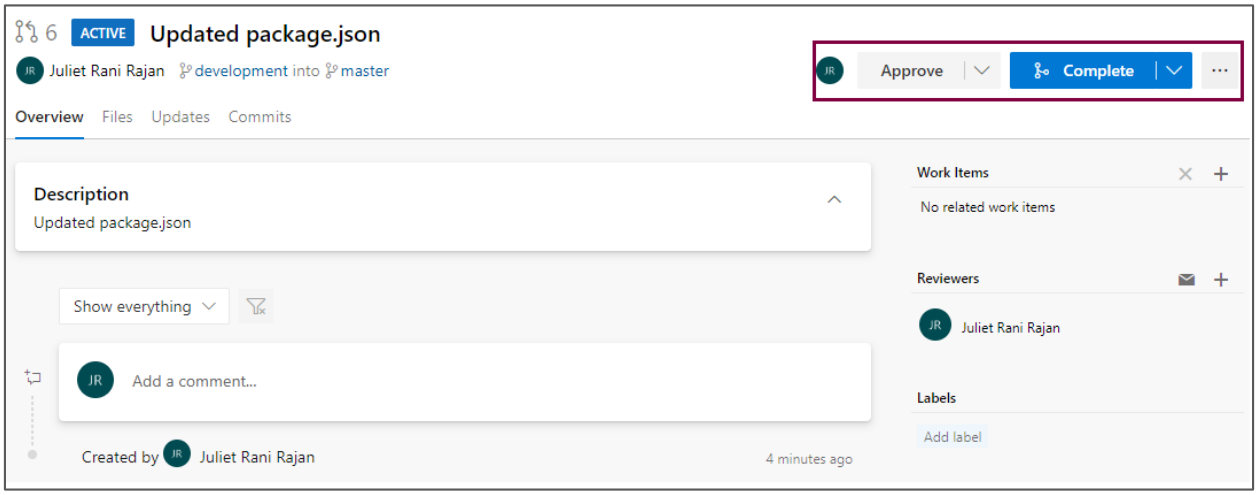


Step 5:

You will see the “Pull Request” that you created and the ones that are submitted to you for approval. Click on the one that is submitted to you for approval.

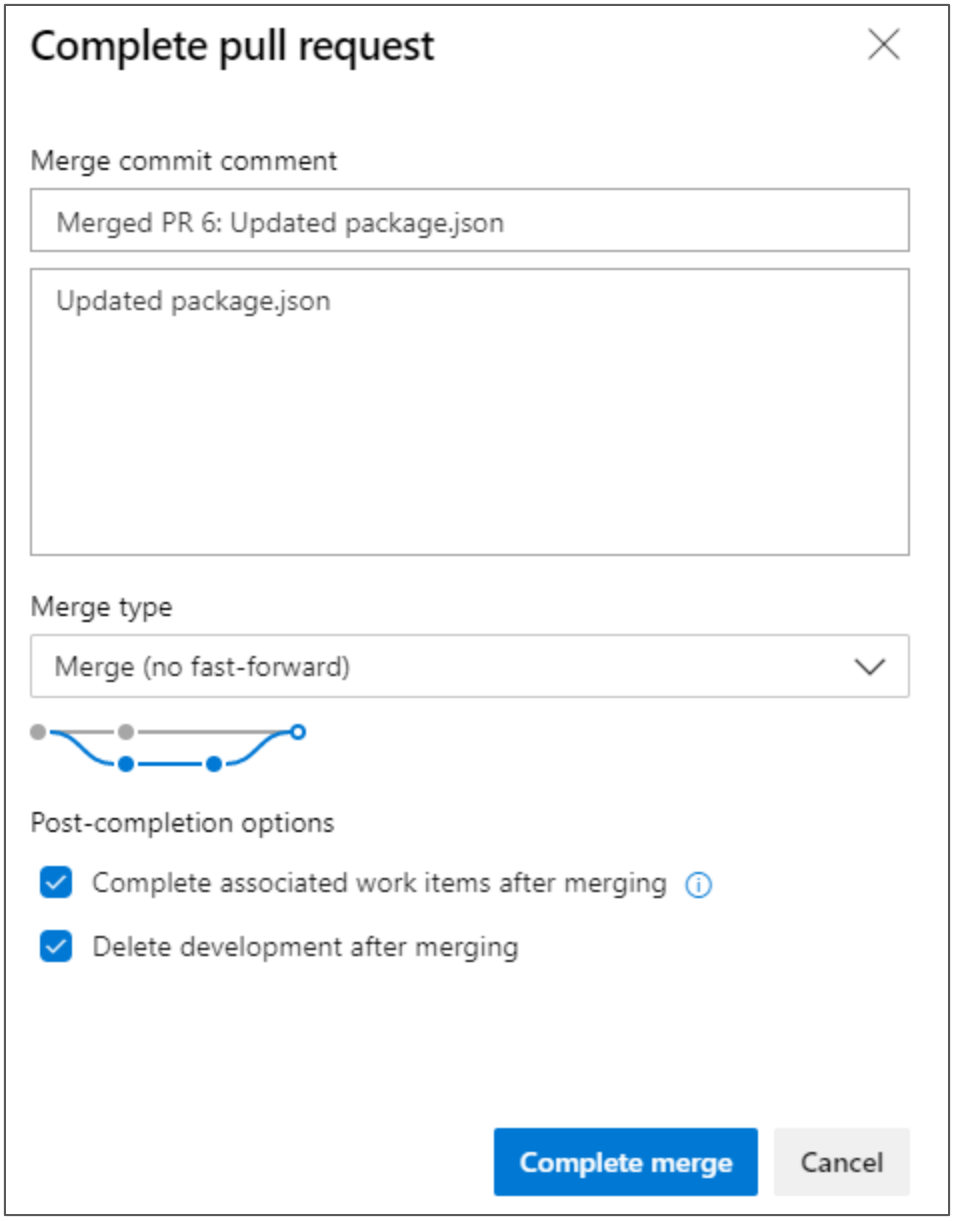


Click on “Approve” and then on “Complete” to approve and merge the changes.

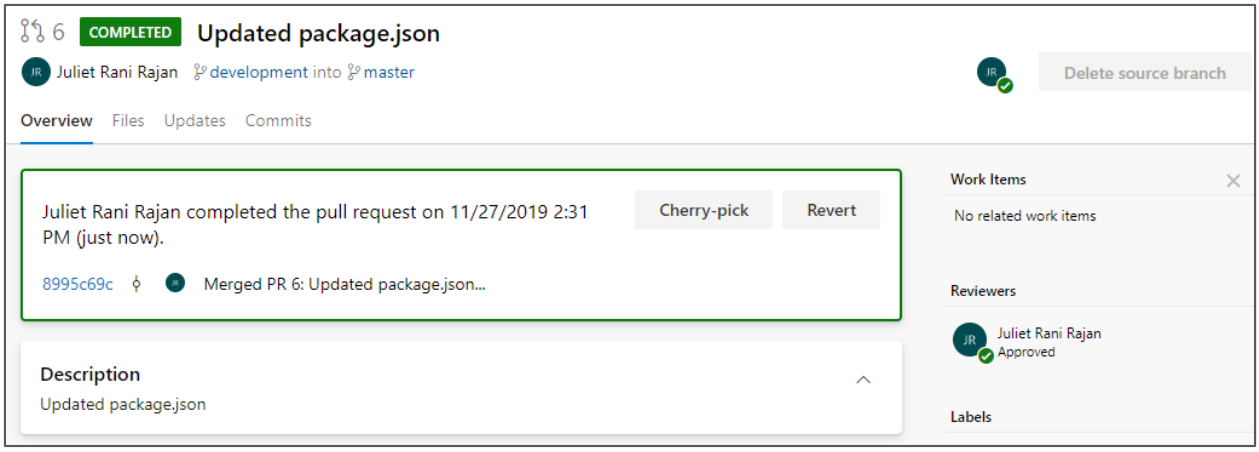


Step 6:

Now complete the “Pull Request”.



You will see this on successful merge.



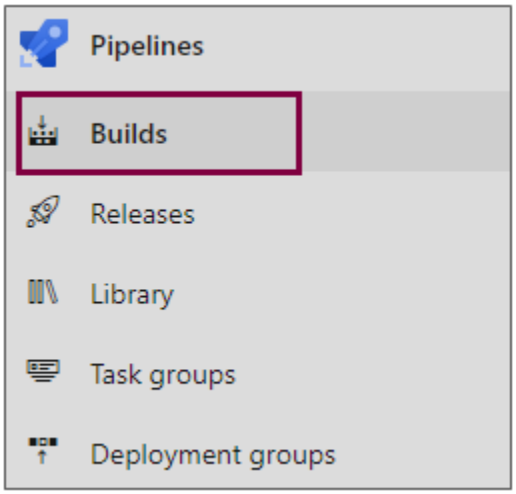
Thus you have seen how to create branches, push changes to branches and finally merge with the master branch.

Azure pipelines

Once the required changes are done to the application, the application can be build and deployed using the Azure Pipelines.

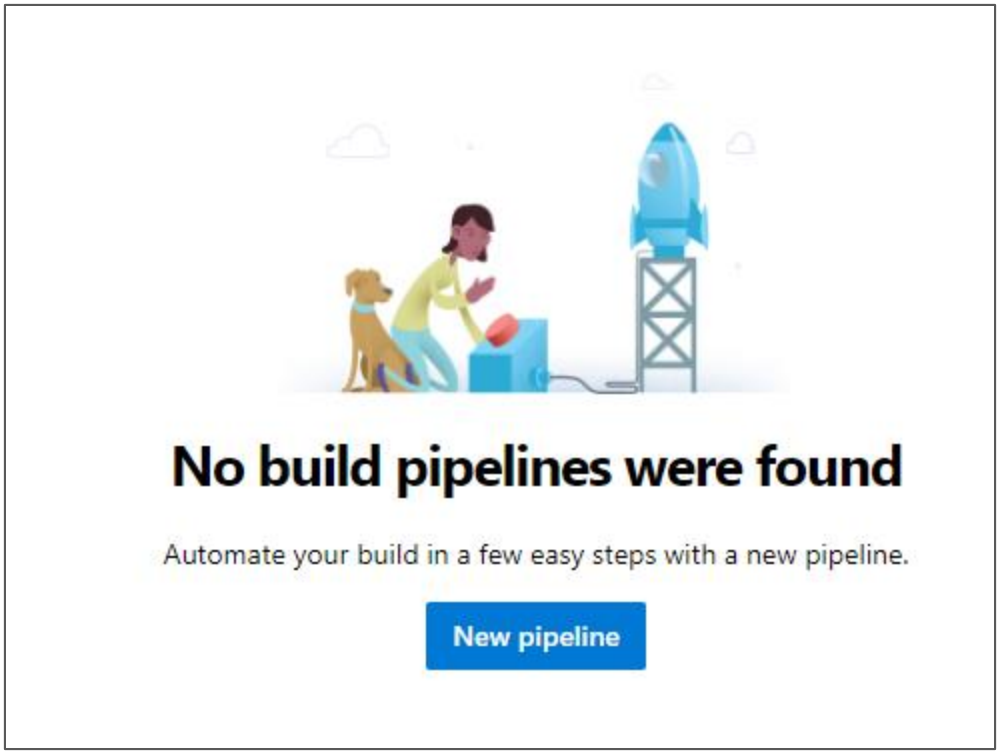
Step 1:

Navigate to builds in "Pipeline".

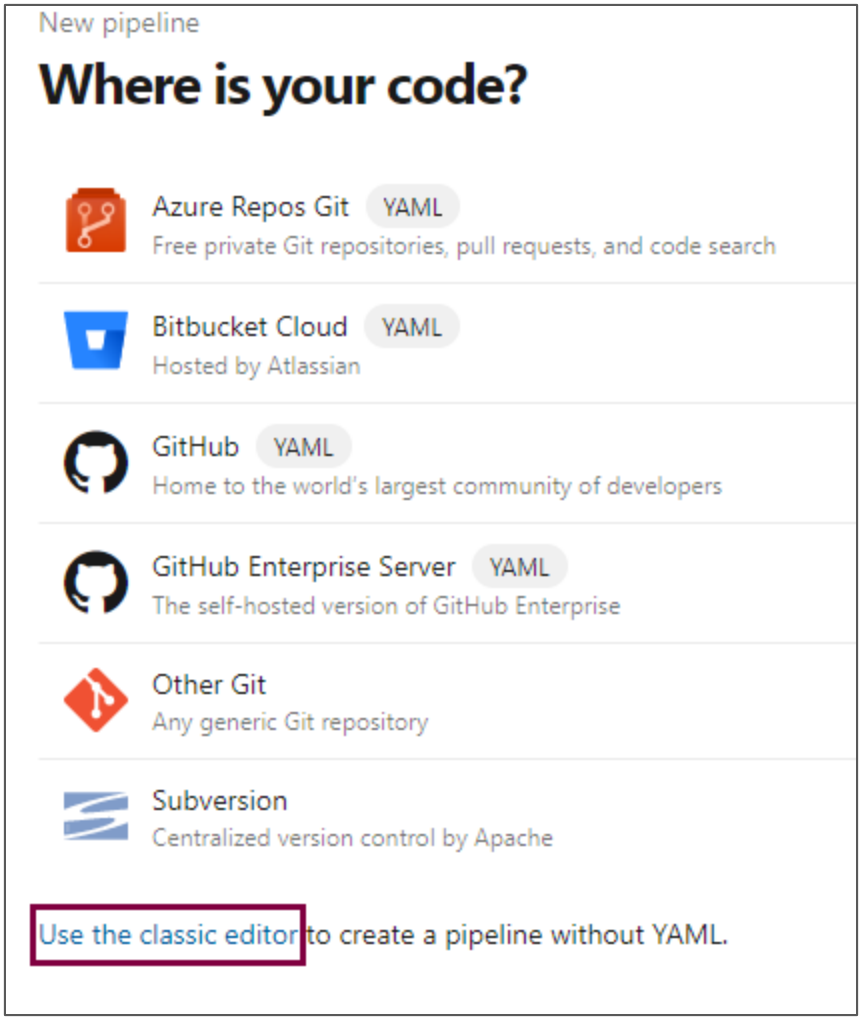


Step 2:

Click on "New Pipleline"

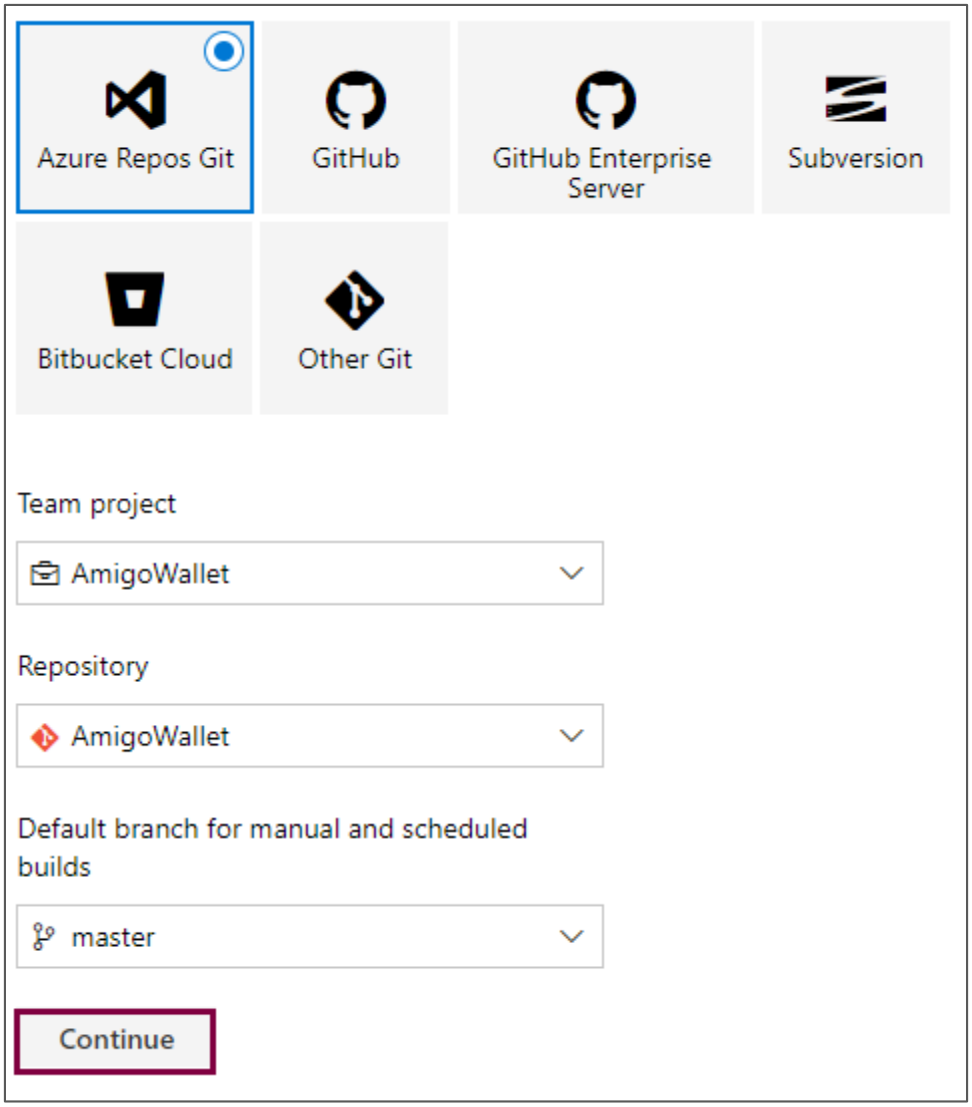


Step 3:
Choose the classis editor to configure the build.



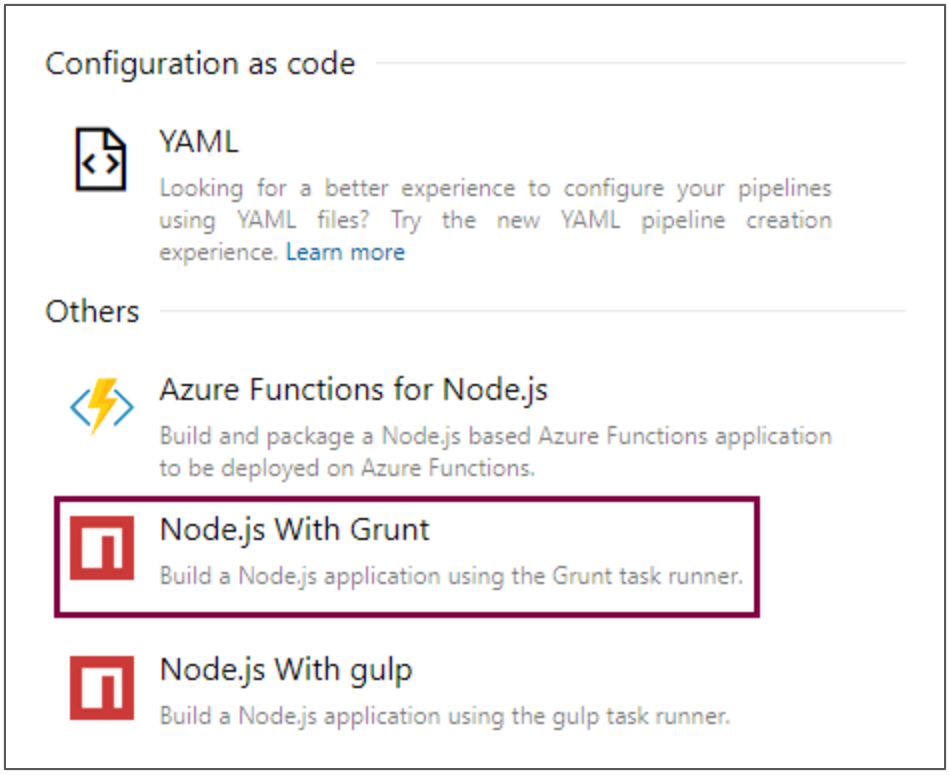
Step 4:

Choose the repository where the application is present.



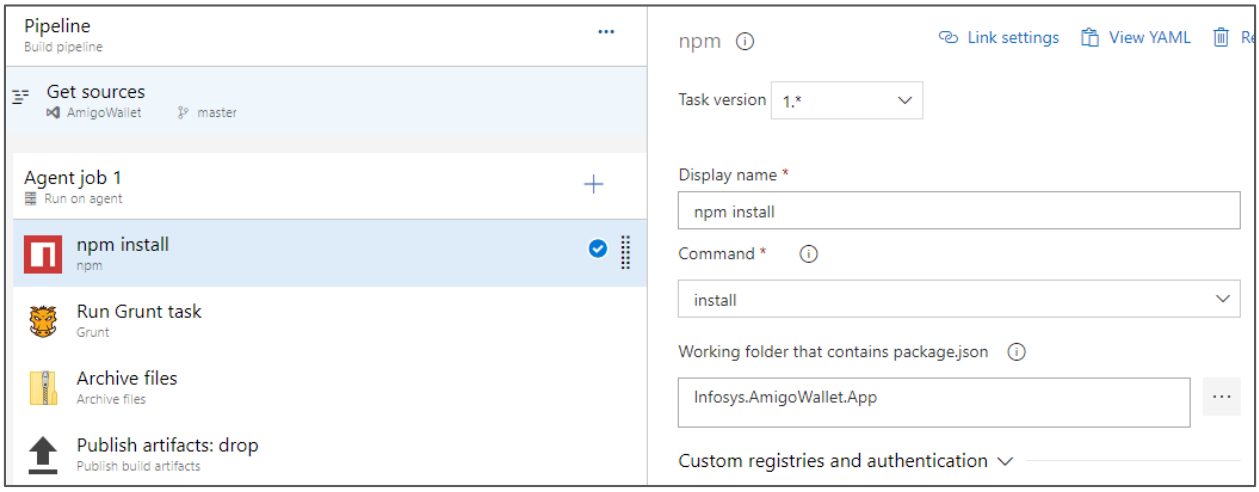
Step 5:

To configure build, choose “Node.js using Grunt” template.


Configuration as code
YAML
Looking for a better experience to configure your pipelines using YAML files? Try the new YAML pipeline creation experience. Learn more
Others
Azure Functions for Node.js
Build and package a Node.js based Azure Functions application to be deployed on Azure Functions.
Node.js With Grunt
Build a Node.js application using the Grunt task runner.
Node.js With gulp
Build a Node.js application using the gulp task runner.

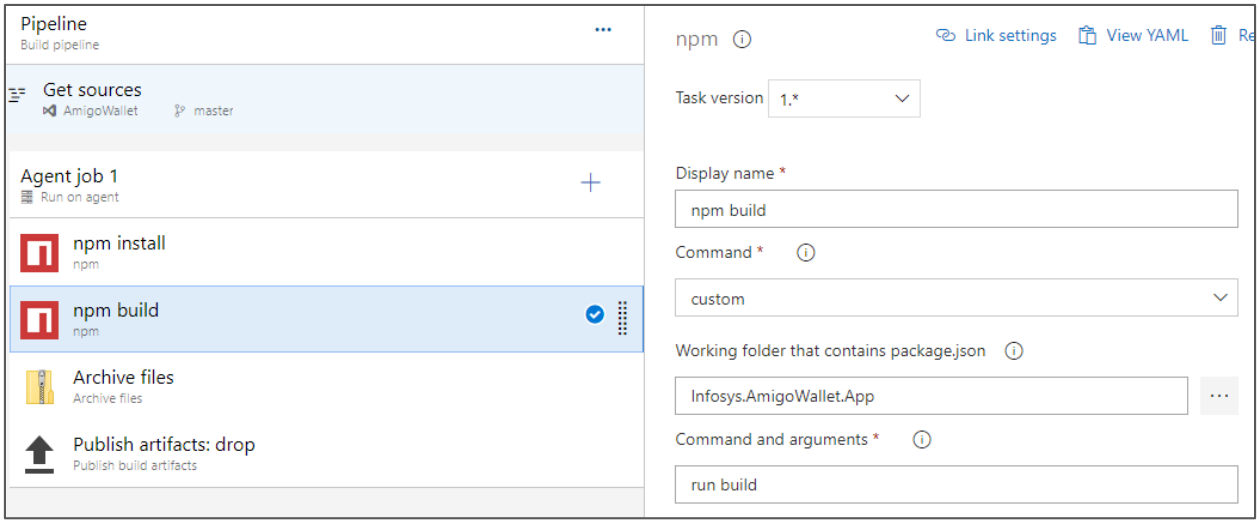
Step 6:

Configure the agents in the build as below.



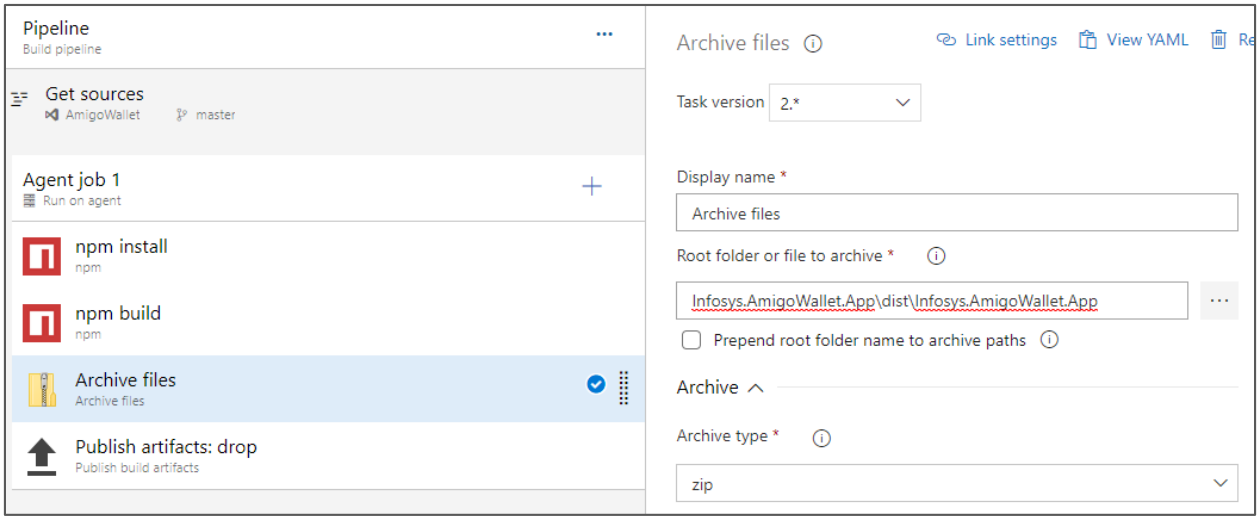
Step 7:

Add one more “npm” agent after the npm install to add the build agent.



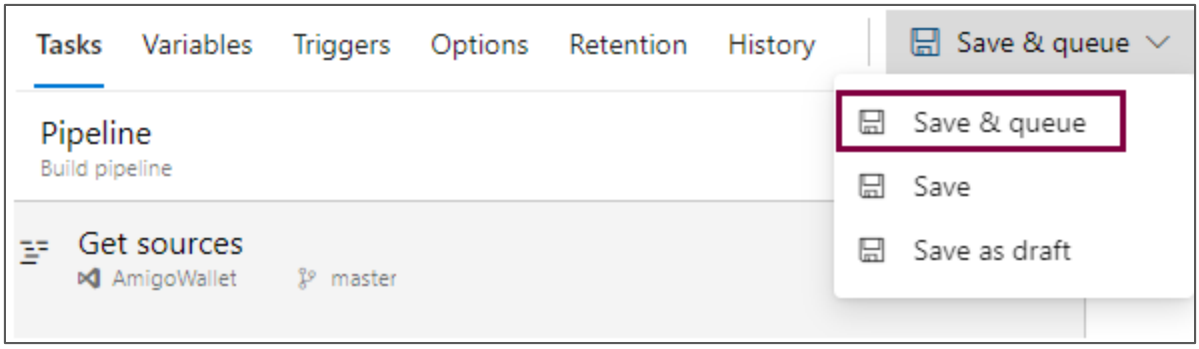
Step 8:

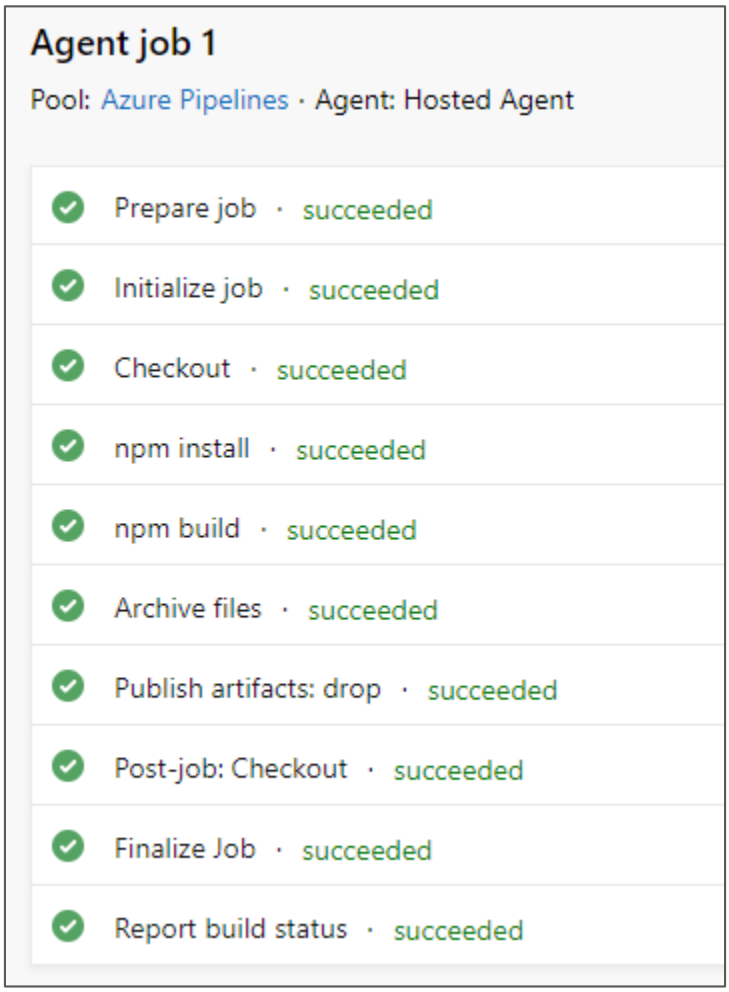
Choose the location to store the resultant artifact.



Step 9:

Now, Save and Queue the build.

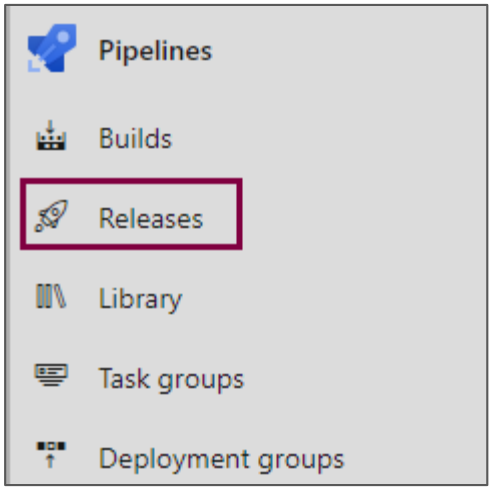



Agent job 1
Pool: Azure Pipelines · Agent: Hosted Agent
Prepare job · succeeded
Initialize job · succeeded
Checkout · succeeded
npm install · succeeded
npm build · succeeded
Archive files · succeeded
Publish artifacts: drop · succeeded
Post-job: Checkout · succeeded
Finalize Job · succeeded
Report build status · succeeded

Once the build is done successfully, you can use the artifacts of build to configure the release.

Step 1:

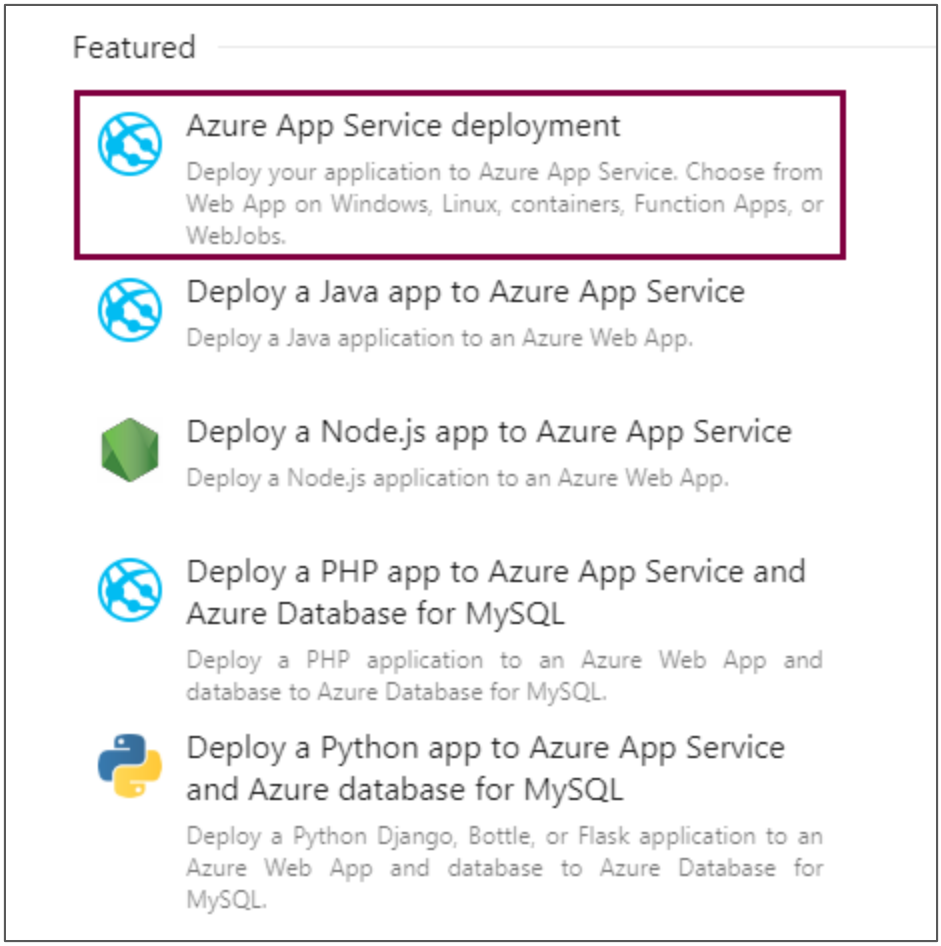
Navigate to pipeline and choose “Releases”.



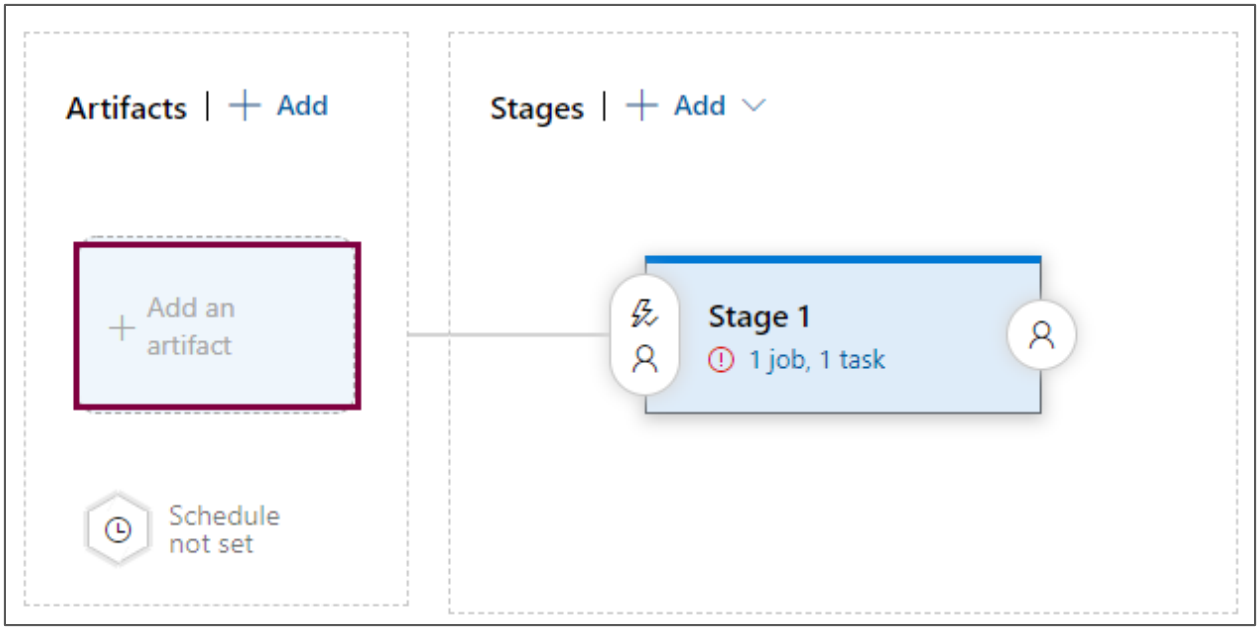
Step 2:
Click on "New Pipeline".

Step 3:
Since the application is going to get deployed to Azure, choose "Azure App Service deployment".

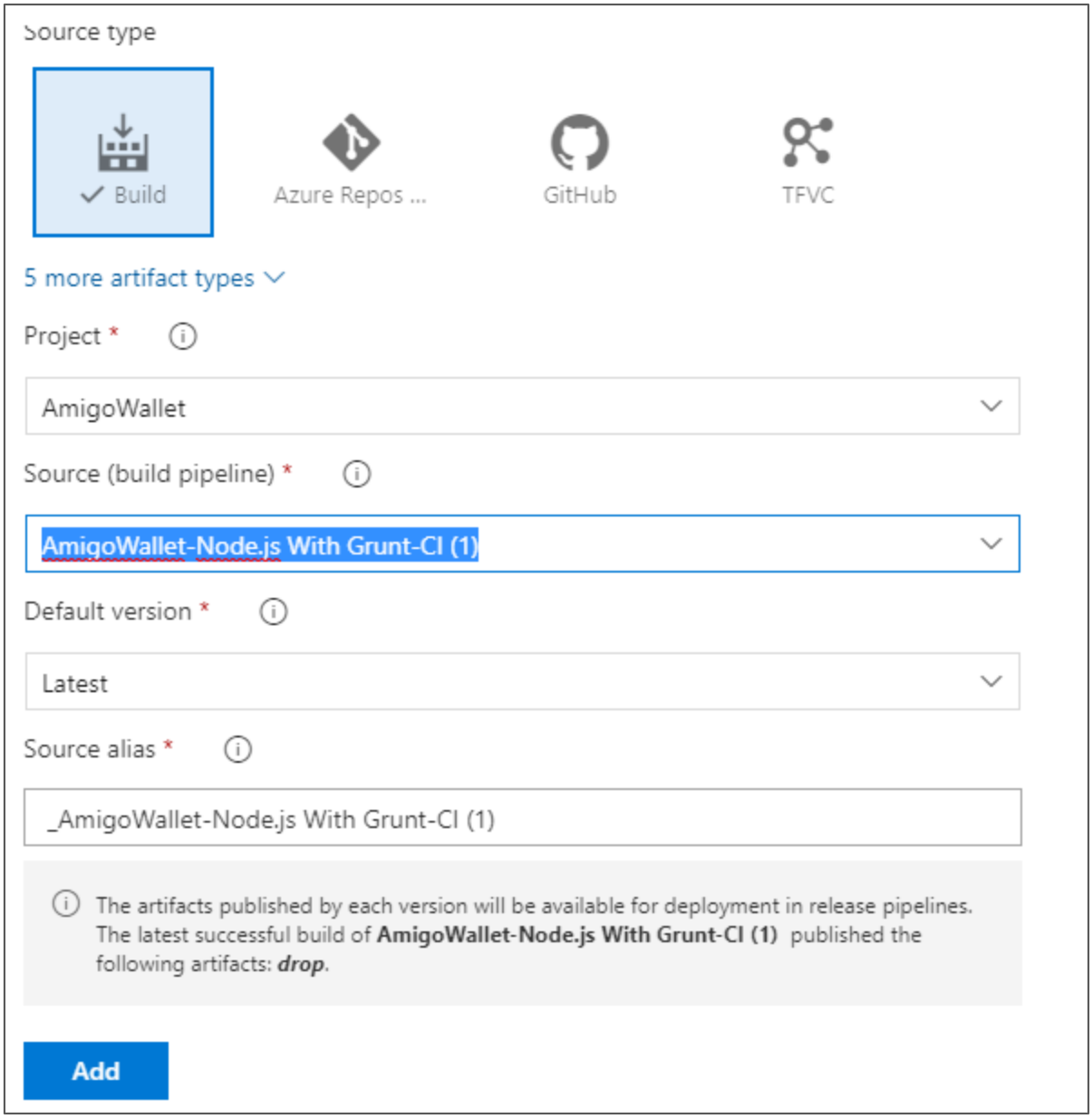
(Note: Make sure you have App Service already created in Azure)


Featured
Azure App Service deployment
Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.
Deploy a Java app to Azure App Service
Deploy a Java application to an Azure Web App.
Deploy a Node.js app to Azure App Service
Deploy a Node.js application to an Azure Web App.
Deploy a PHP app to Azure App Service and Azure Database for MySQL
Deploy a PHP application to an Azure Web App and database to Azure Database for MySQL.
Deploy a Python app to Azure App Service and Azure database for MySQL
Deploy a Python Django, Bottle, or Flask application to an Azure Web App and database to Azure Database for MySQL.

Step 4:
Add the corresponding "Artifact" by clicking on "Add an artifact".

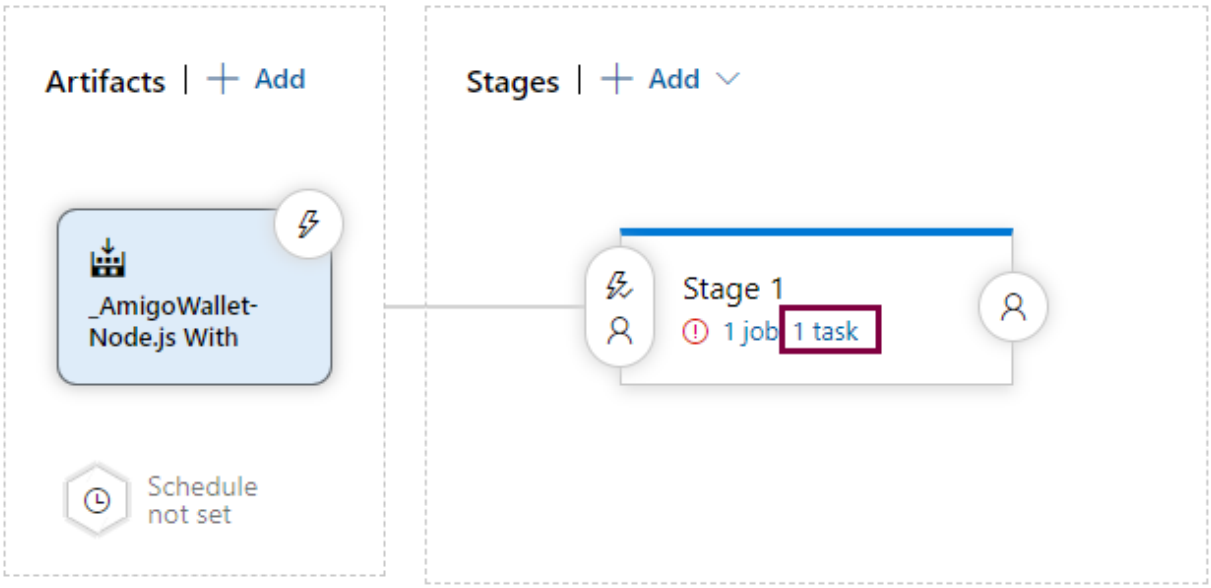


Step 5:
Choose the build artifacts that was produced by the build pipeline.

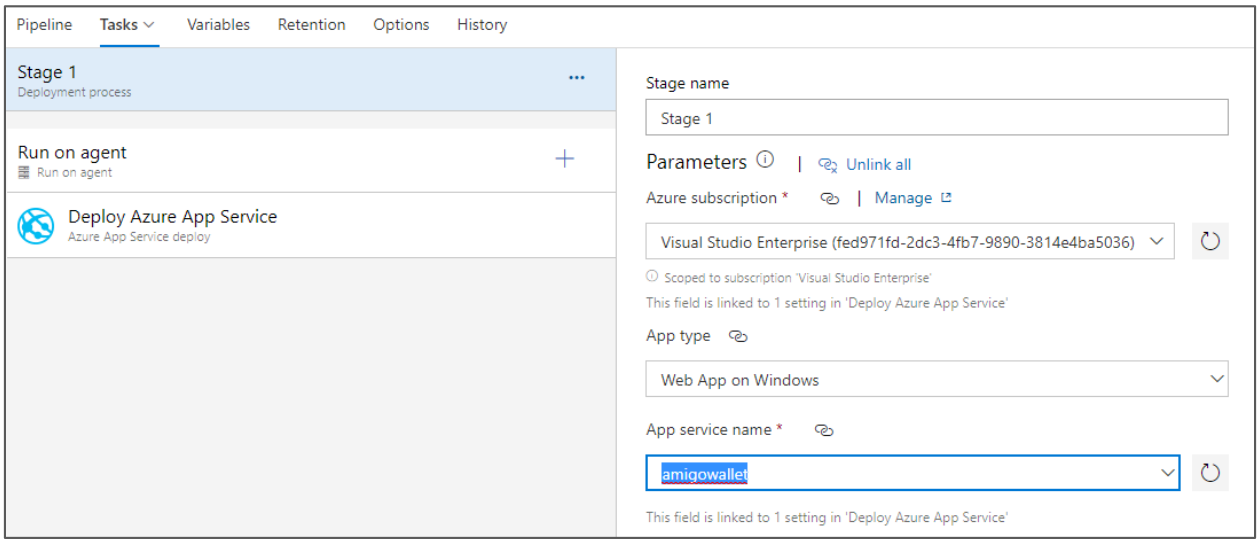


Step 6:

Now add the Stage details.



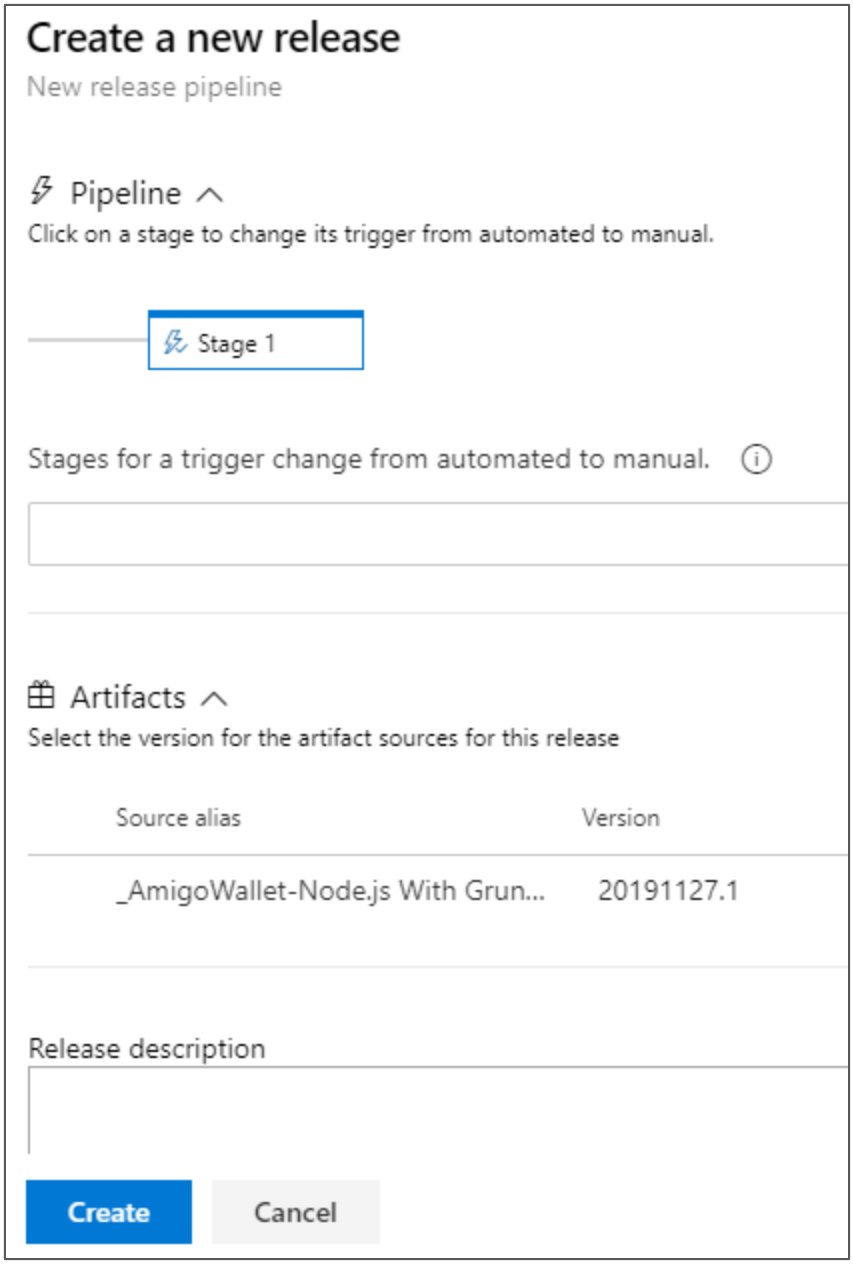
Provide the details of the Web App service where this application is going to get deployed.



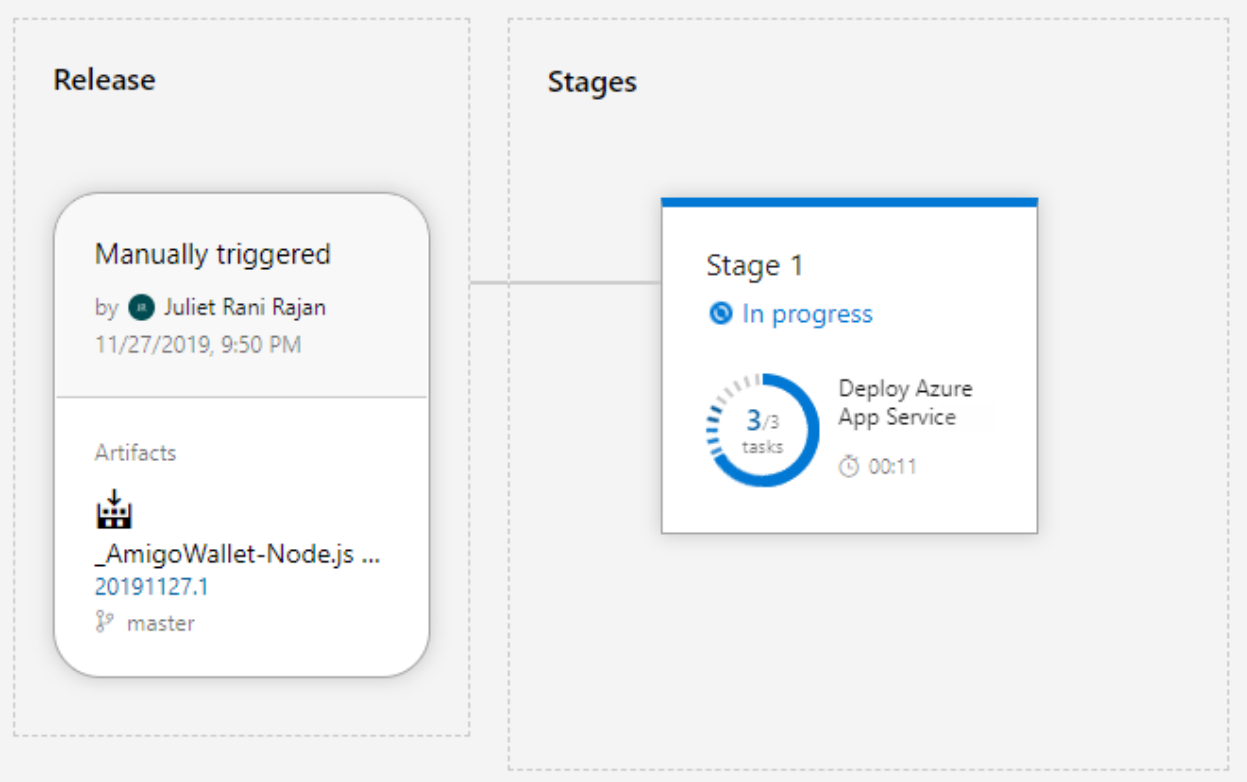
Step 7:

Click on "Create Release" once the configurations are update.

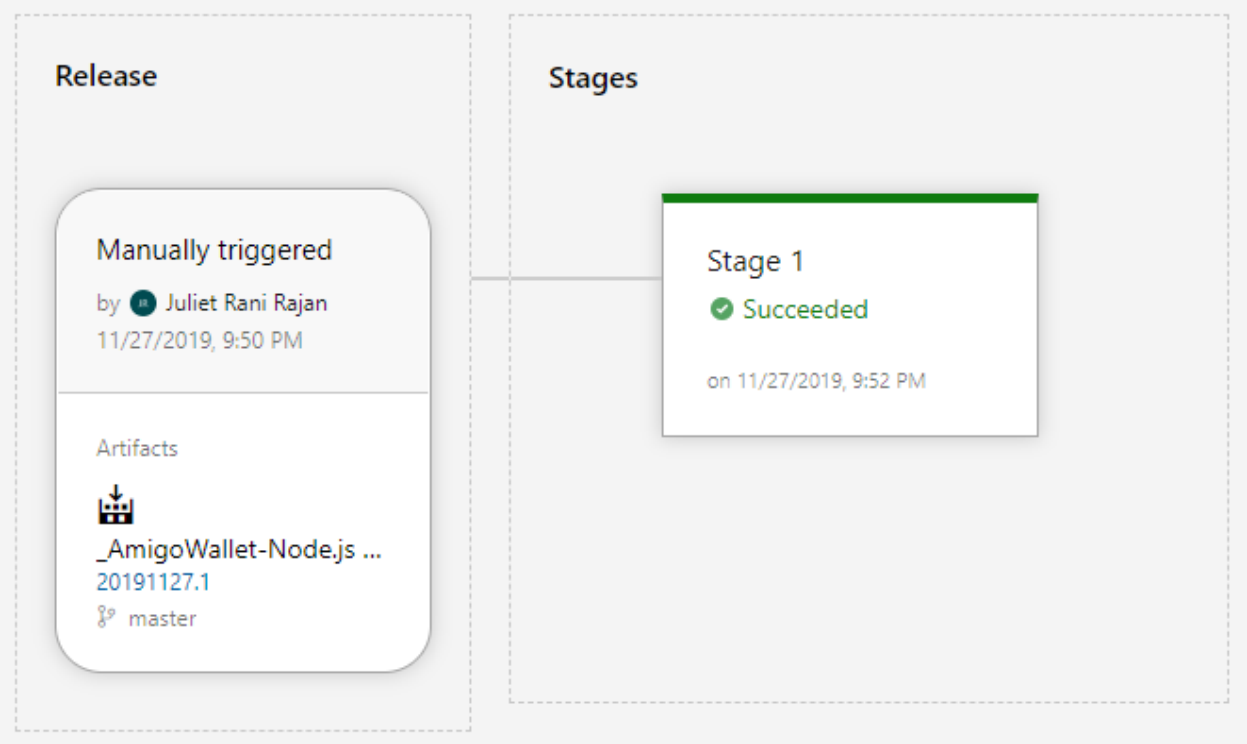
Step 8:
Click on "Create" to start the deployment.



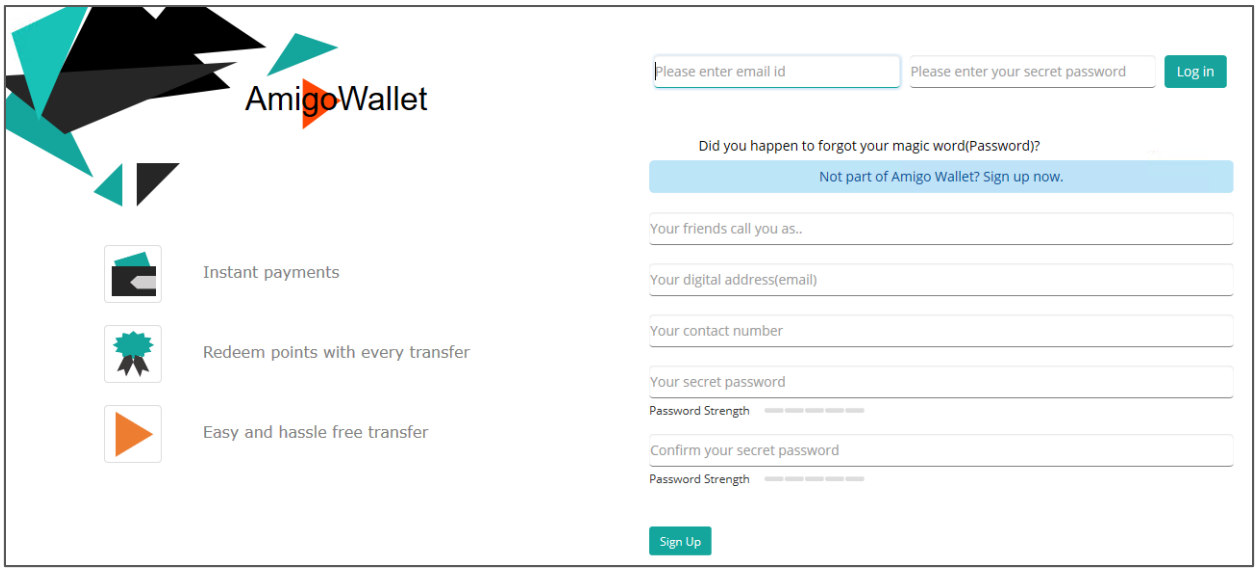
You can see the status of your deployment.



You will see the below on successful deployment.



Now browse the application, you will see the below.



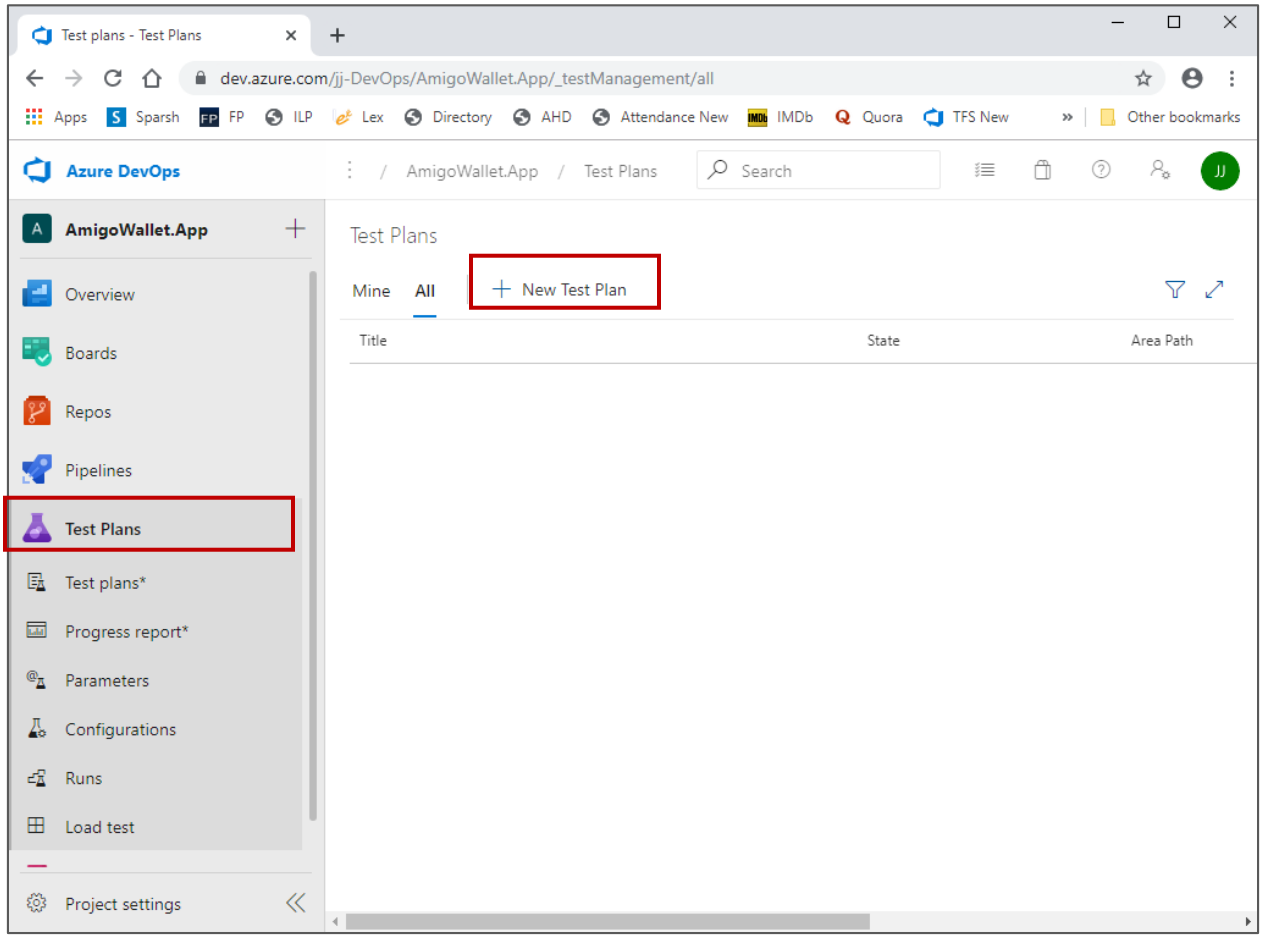
Azure Test Plan

Test plans are used to track manual testing for sprints or milestones. Using test plans we can see when the testing for a specific sprint or milestone is complete. Test plans group test suites and individual test cases together and Test suites group test cases into separate testing scenarios within a single test plan. Grouping test cases makes it easier to see which scenarios are complete. Test cases validate individual parts of your code or app deployment. You can ensure your code works correctly, has no errors, and meets business and customer requirements. You can add individual test cases to a test plan without creating a test suite, if you wish. More than one test suite or test plan can refer to a test case.

Let's see how to create a Test plan and Test cases.

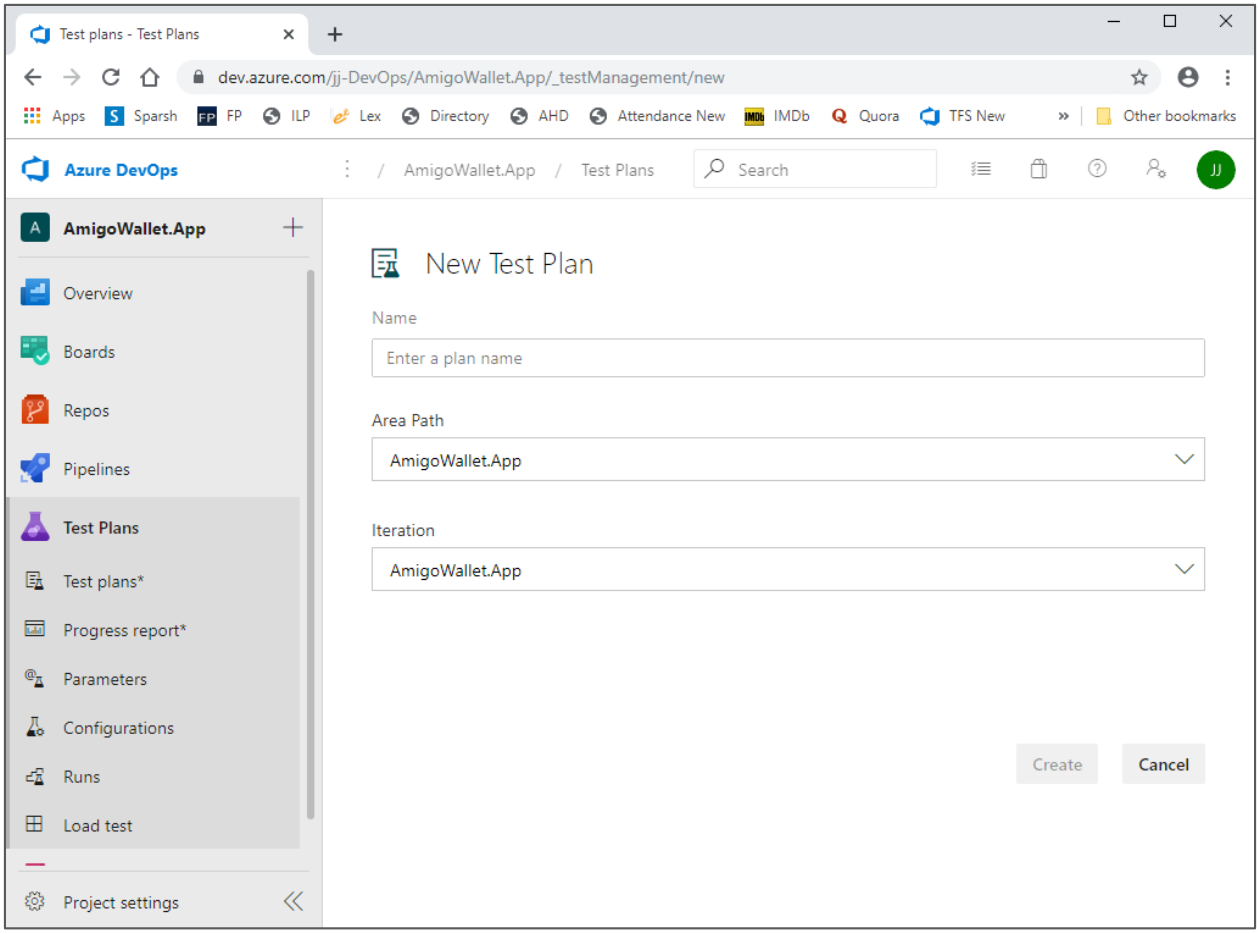
Step 1:

Go to DevOps homepage and click on **Test Plans** -> New Test Plan.



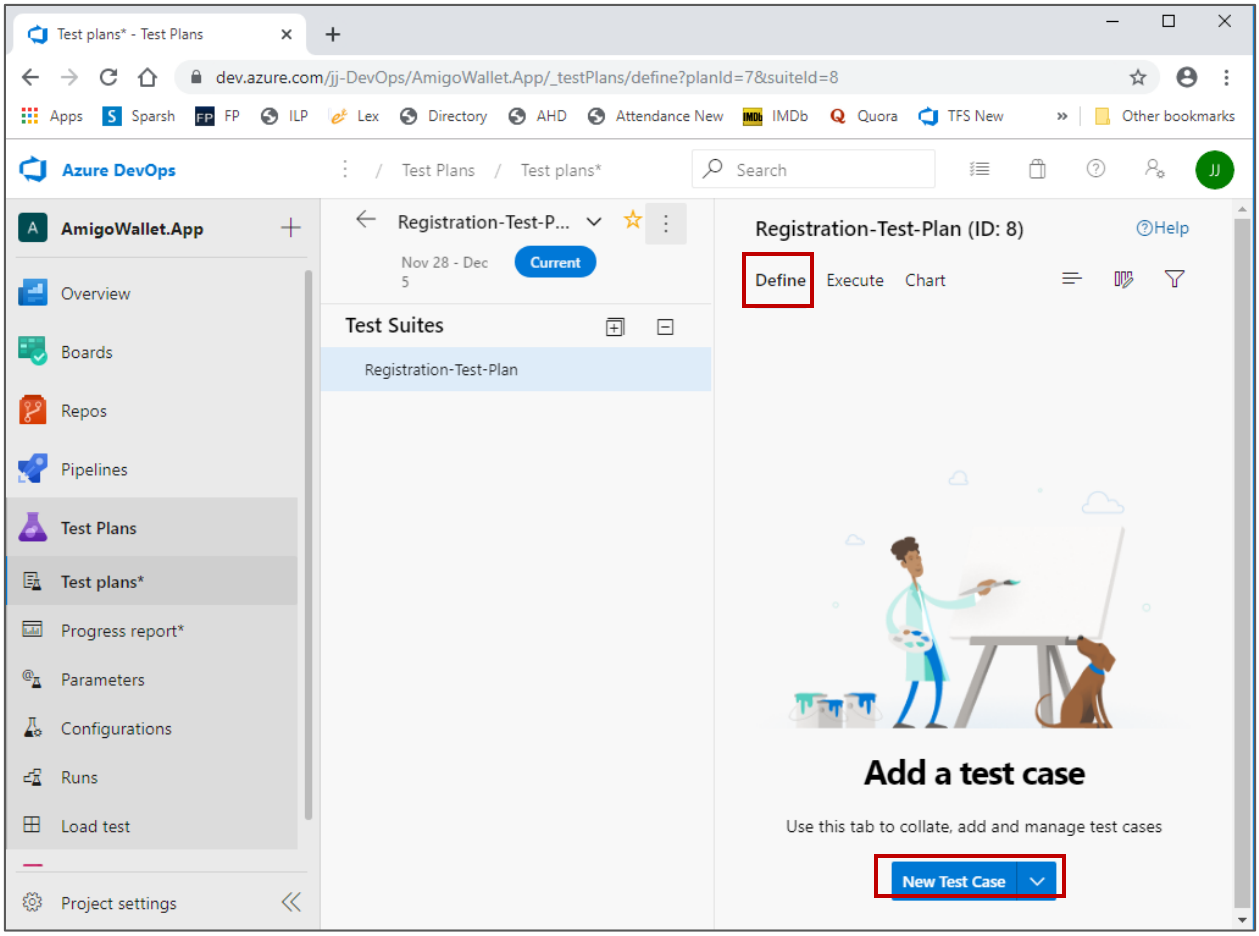
Step 2:

For creating a test plan, provide the required details such as name, Area path, and Iteration and click on **Create** button. In the iteration, we can select the corresponding sprint where the test plan is a part of.



Step 3:

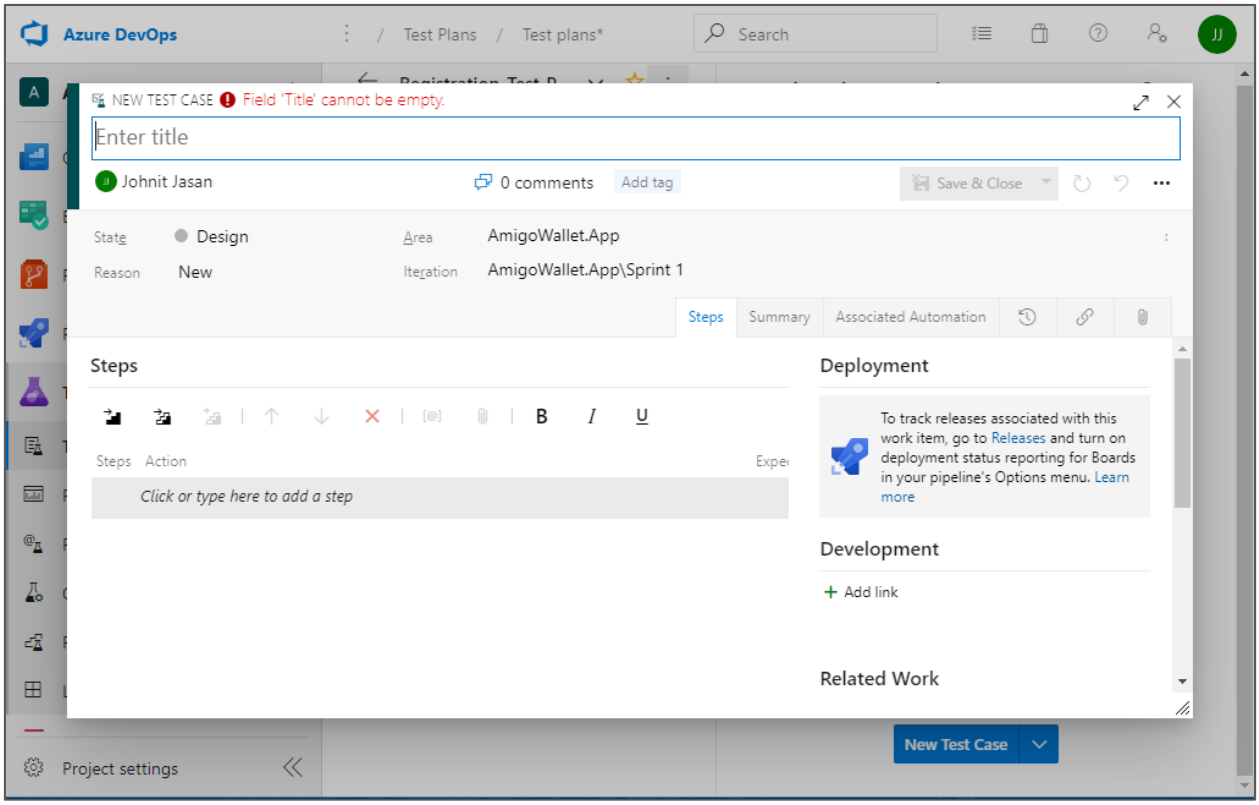
Now that we have created a Test Plan, we can add test cases to this test plan. For doing this, select the **Define** tab inside the Test plan and click on **New test case.**



Step 4:

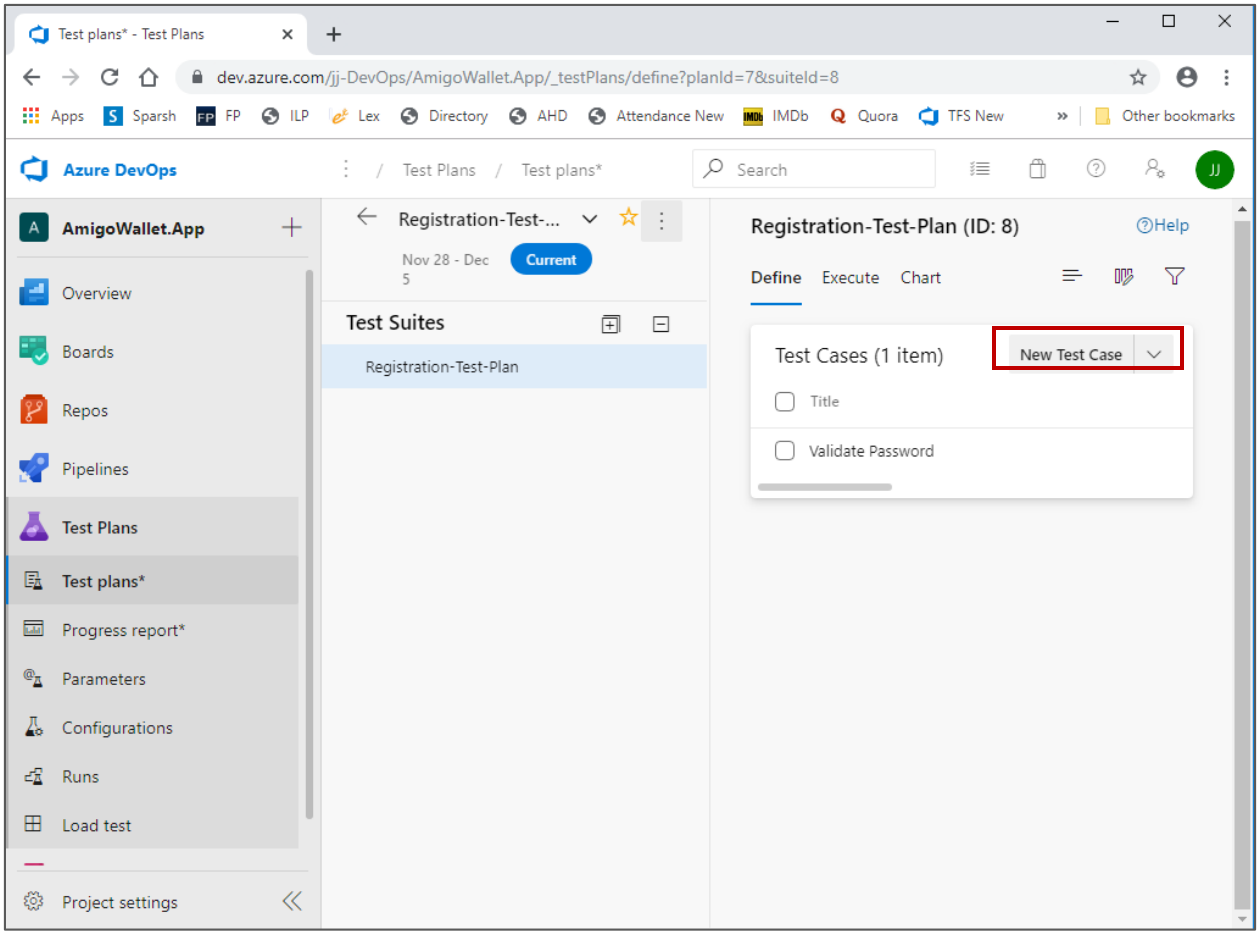
Provide a Title to the Test case and assign it to a user. You can provide description, steps required for the test, priority, comments etc. also. There's also features for linking the test case to a branch of the project or a particular work item in the board.

Click on **Save & Close**.



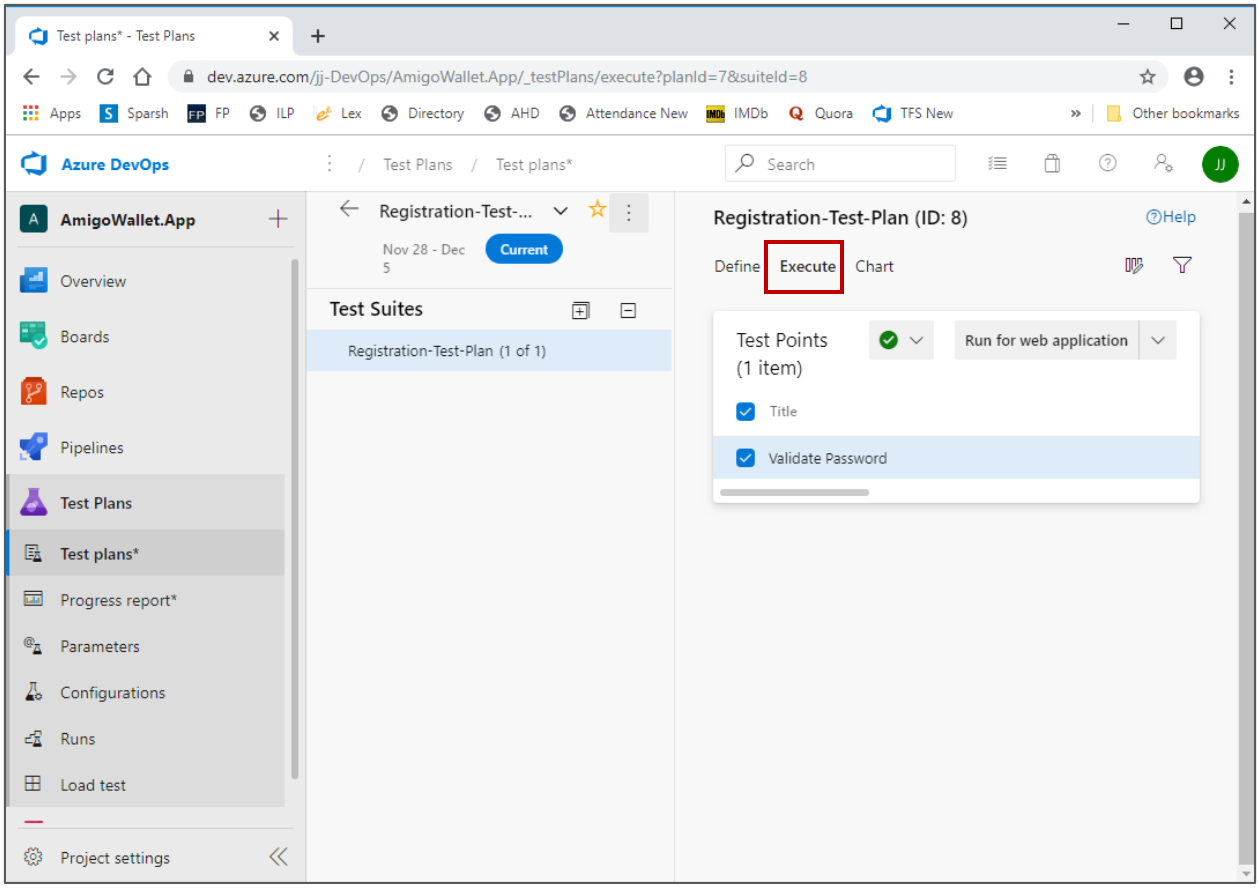
Step 5:

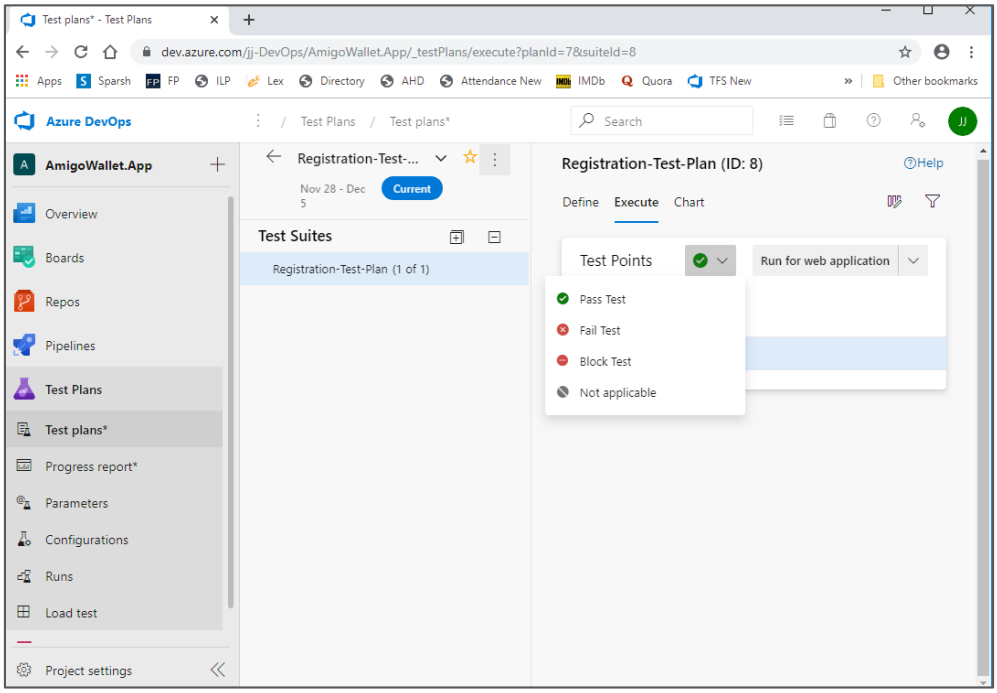
You can create more test cases by clicking on **New Test case.**



Step 6:

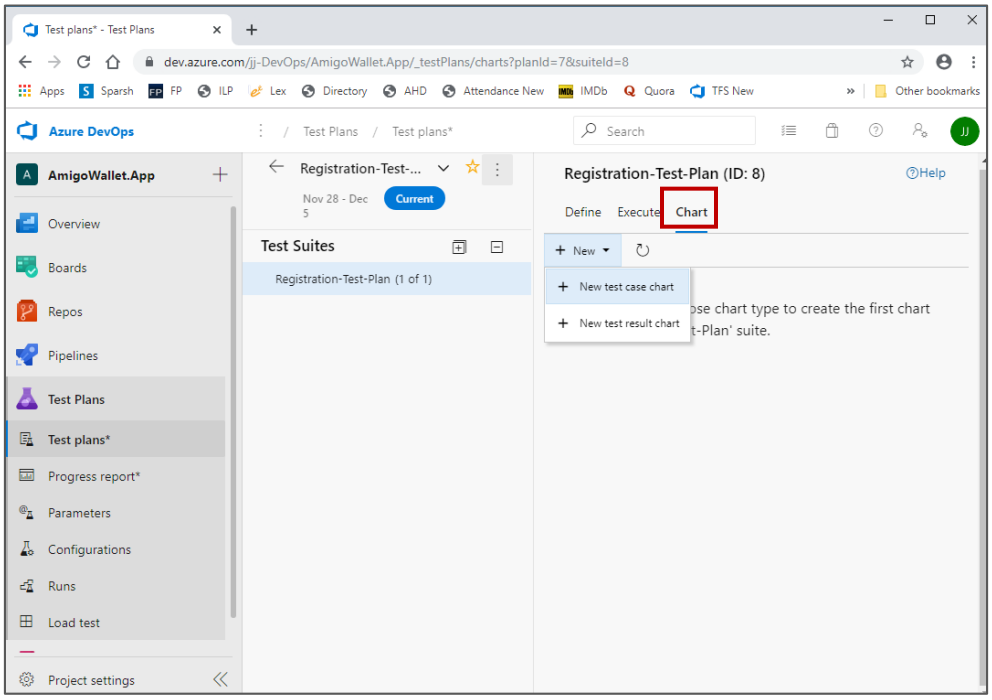
We can execute the test cases or set different statuses for the test cases from the **Execute** tab.


Test plans* - Test Plans
dev.azure.com/jj-DevOps/AmigoWallet.App/_testPlans/execute?planId=7&suiteId=8
Azure DevOps
Test Plans / Test plans*
Search
AmigoWallet.App
Overview
Boards
Repos
Pipelines
Test Plans
Test plans*
Progress report*
Parameters
Configurations
Runs
Load test
Project settings
Registration-Test-...
Nov 28 - Dec 5
Current
Test Suites
Registration-Test-Plan (1 of 1)
Registration-Test-Plan (ID: 8)
Help
Define
Execute
Chart
Test Points (1 item)
Run for web application
Title
Validate Password

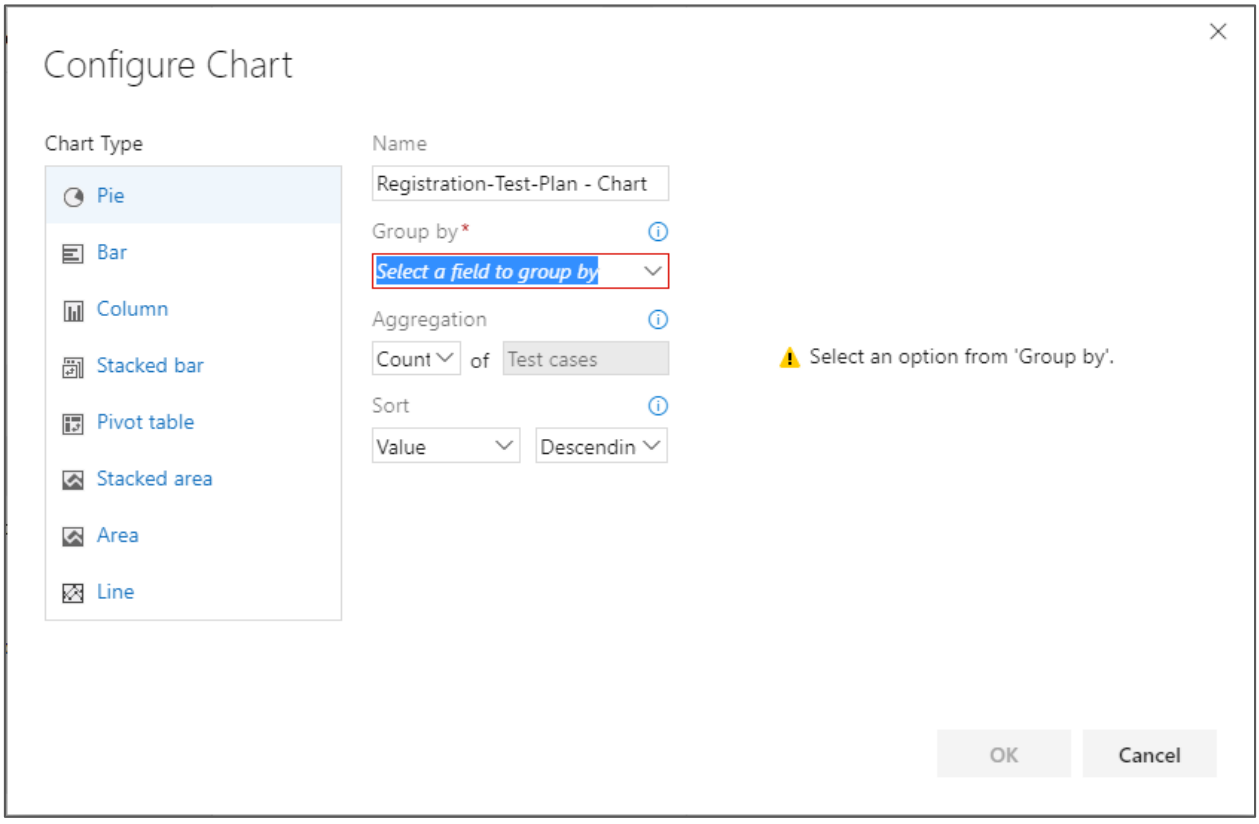

Test plans* - Test Plans
dev.azure.com/jj-DevOps/AmigoWallet.App/_testPlans/execute?planId=7&suiteId=8
Azure DevOps
Test Plans / Test plans*
Search
AmigoWallet.App
Overview
Boards
Repos
Pipelines
Test Plans
Test plans*
Progress report*
Parameters
Configurations
Runs
Load test
Project settings
Registration-Test-...
Nov 28 - Dec 5
Current
Test Suites
Registration-Test-Plan (1 of 1)
Registration-Test-Plan (ID: 8)
Help
Define
Execute
Chart
Test Points
Run for web application
Pass Test
Fail Test
Block Test
Not applicable

Step 7:

We can also create charts based on the test cases. Go to the **Chart** -> New -> New Test case chart.



Provide the required data to generate the chart.

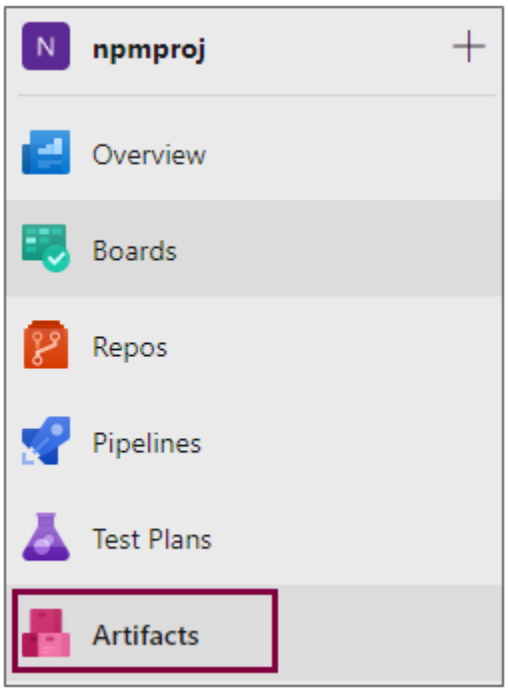


Azure Artifacts

Before you get started with this demo, ensure that you have created the appropriate libraries and the libraries are pushed to the "Azure Repos". Also, the application should be build using the "Azure Pipelines".

Navigate to https://dev.azure.com.

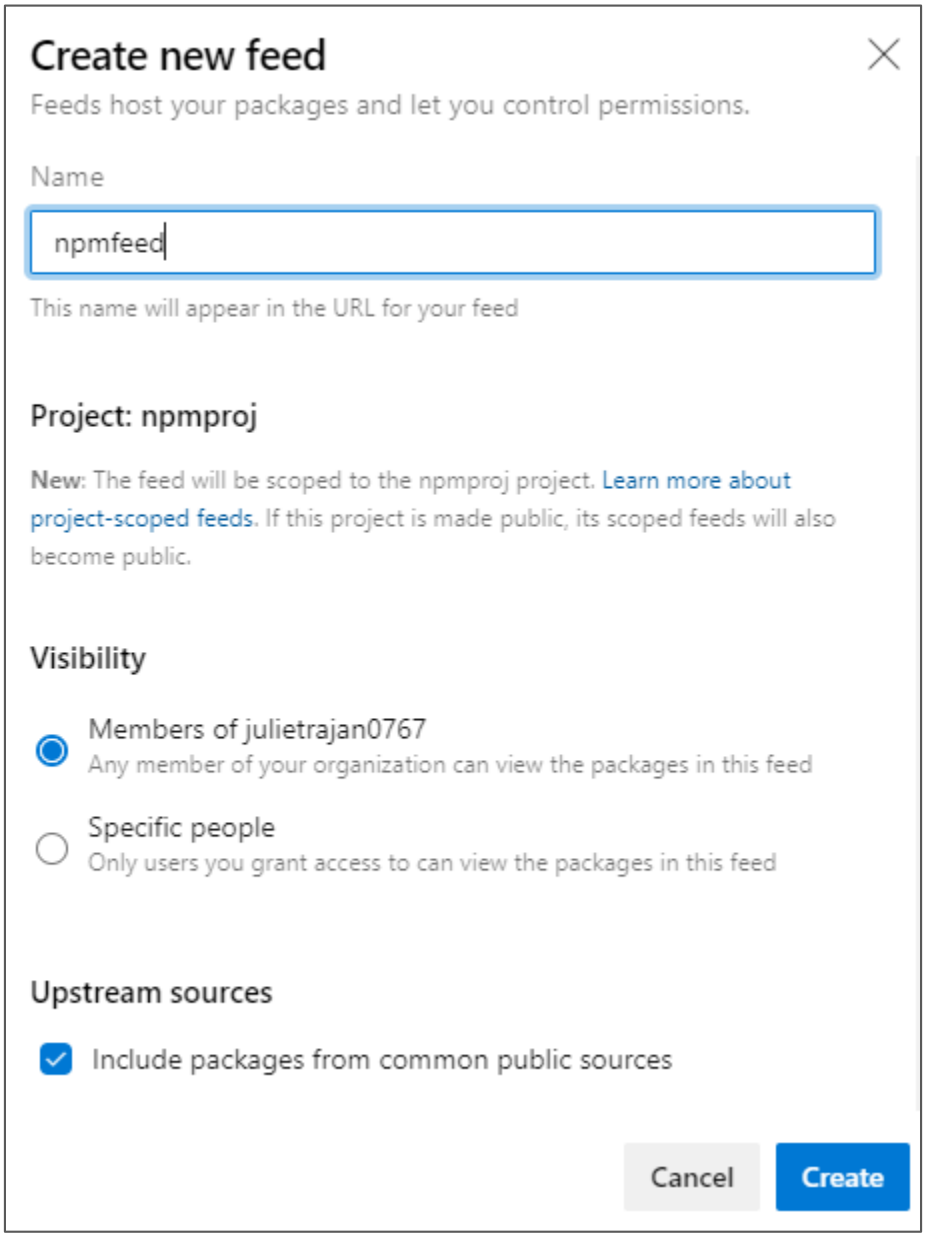
Step 1:
Click on "Artifacts".



Step 2:
You will see the an option to create a feed. Click on "Create feed" to create a new feed.

Step 3:

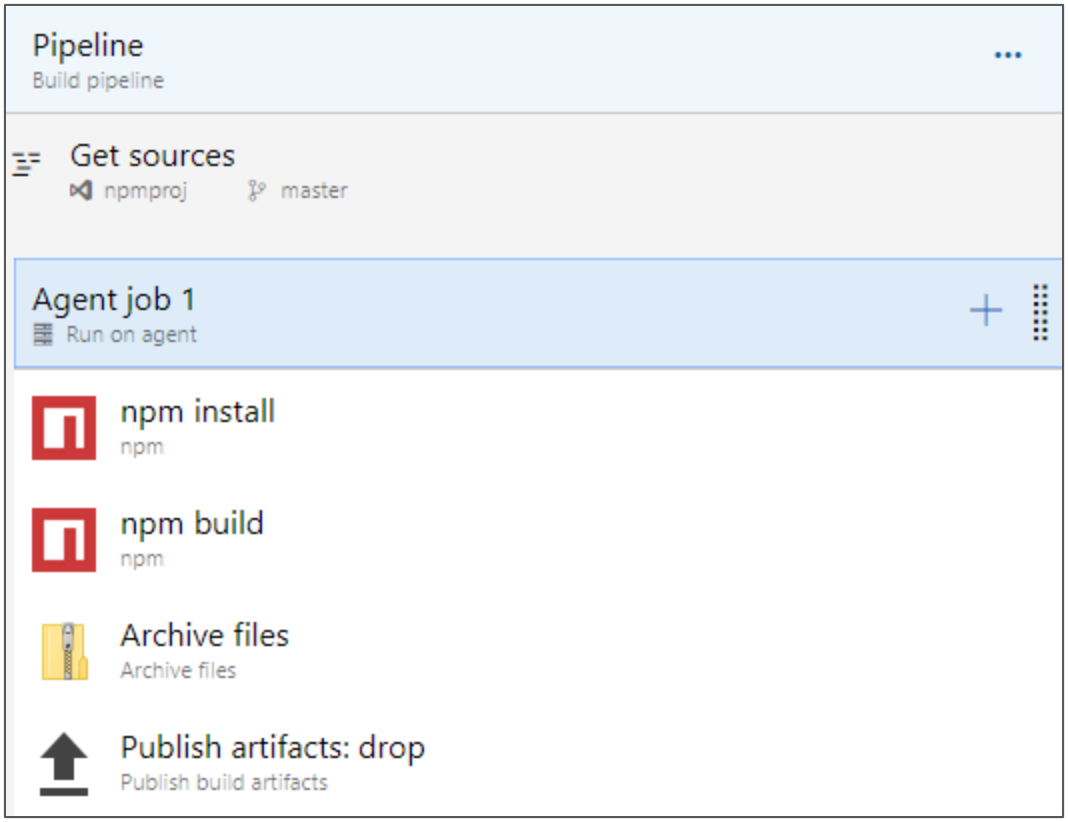
Create a new feed “npmfeed” and give the visibility as mentioned below.


Create new feed
Feeds host your packages and let you control permissions.
Name
npmfeed
This name will appear in the URL for your feed
Project: npmproj
New: The feed will be scoped to the npmproj project. Learn more about project-scoped feeds. If this project is made public, its scoped feeds will also become public.
Visibility
Members of julietrajan0767
Any member of your organization can view the packages in this feed
Specific people
Only users you grant access to can view the packages in this feed
Upstream sources
Include packages from common public sources
Cancel
Create

Step 4:

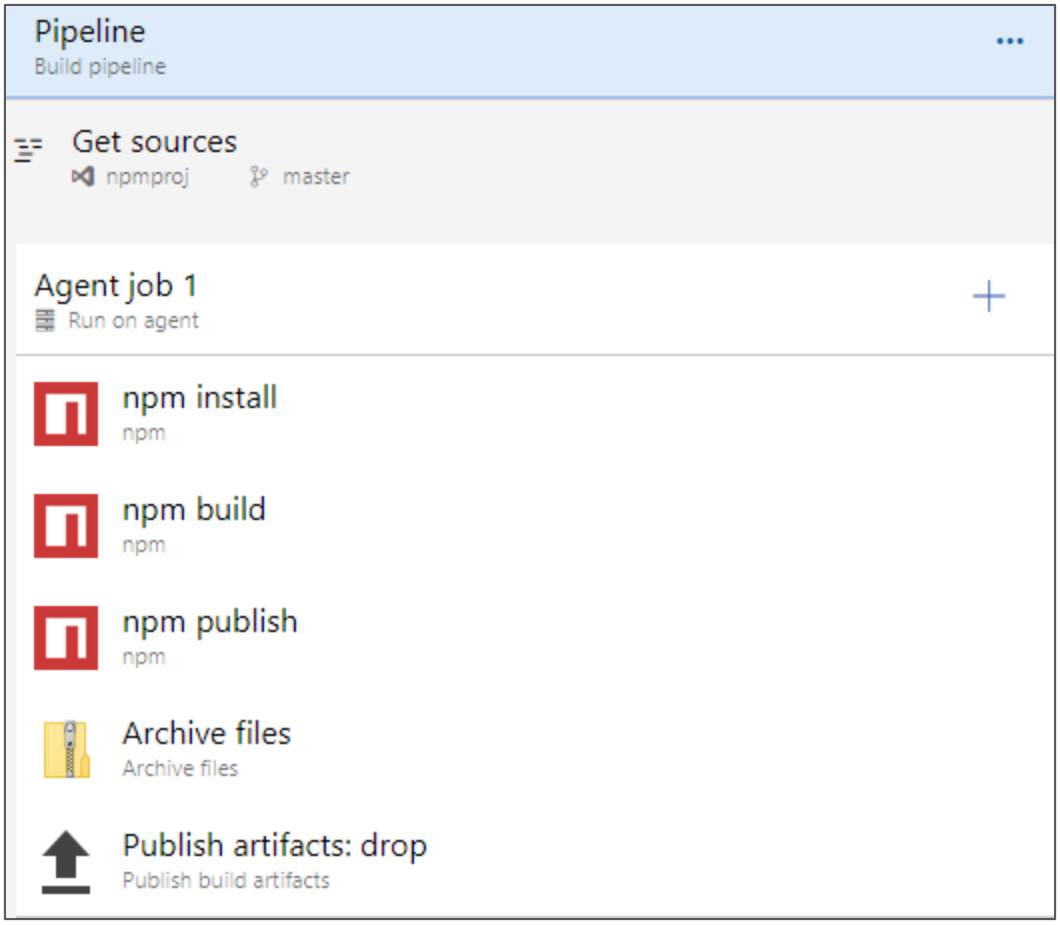
Once a new feed is created.

Open the already existing build pipeline and edit the build. The already existing build pipeline looks as below.



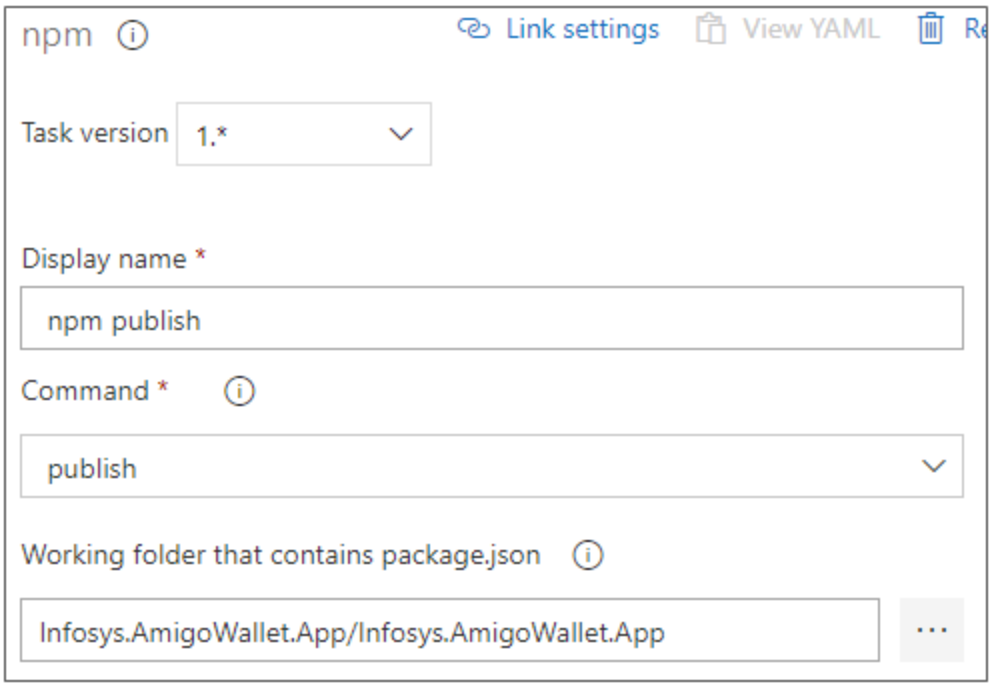
Step 5:

Add one more “npm” agent after the “npm build” and name it as “npm publish”.

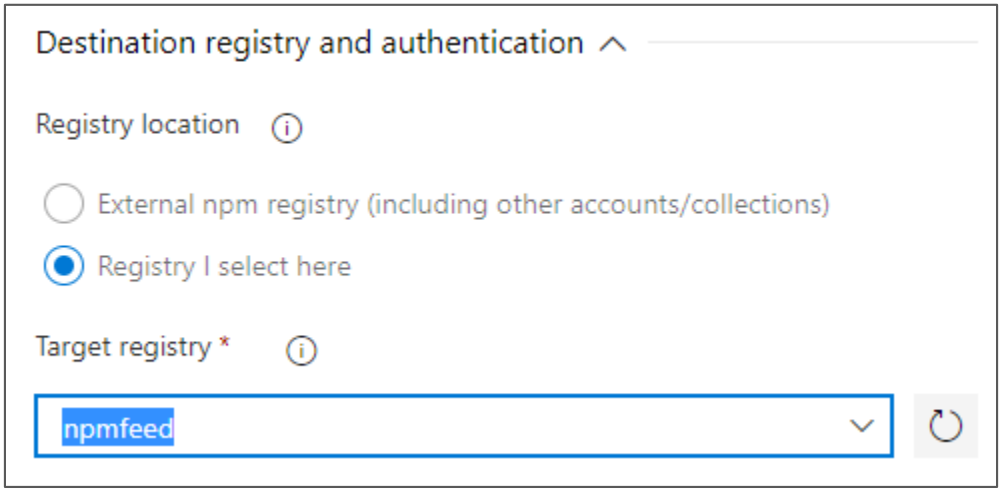


Step 6:

Click on "npm publish" and provide the details as below.

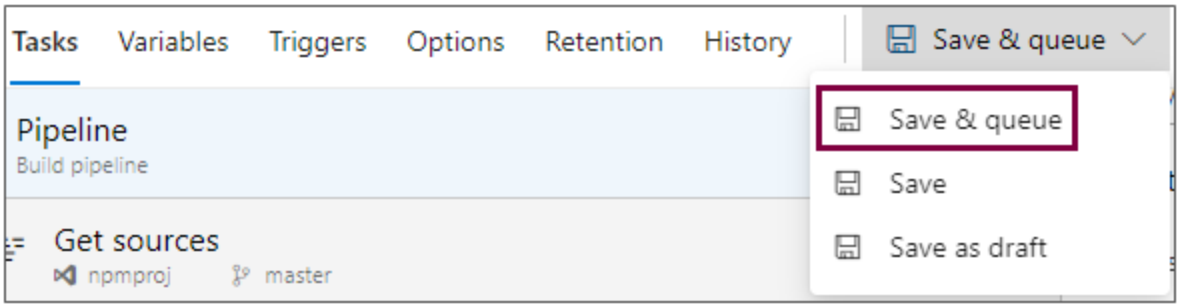


Choose the "npmfeed" as the target registry. This is the feed that was created earlier.



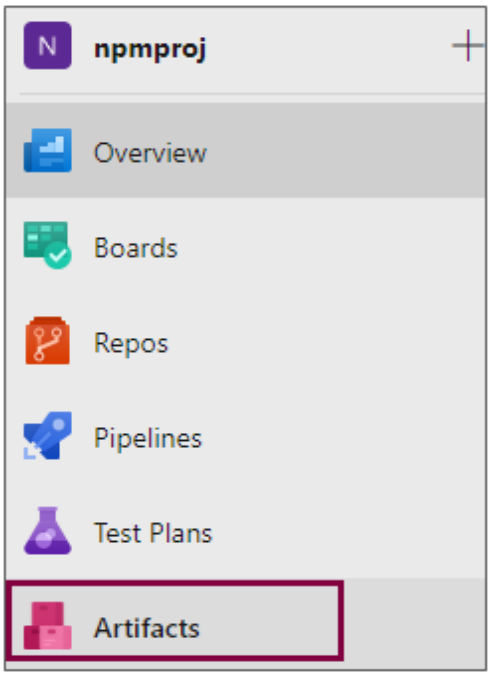
Step 7:

Now "Save and Queue" the build.

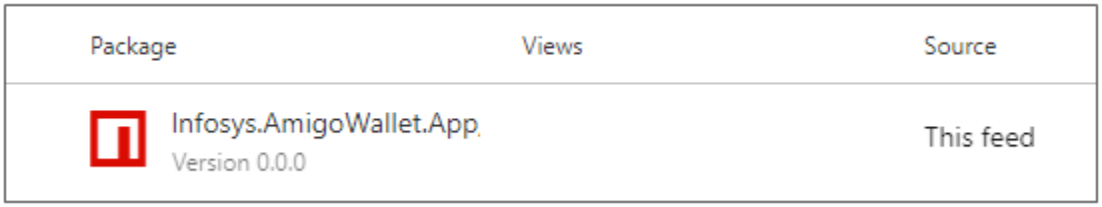


Step 8:

Once the build is completed successfully, navigate to the "Artifacts".



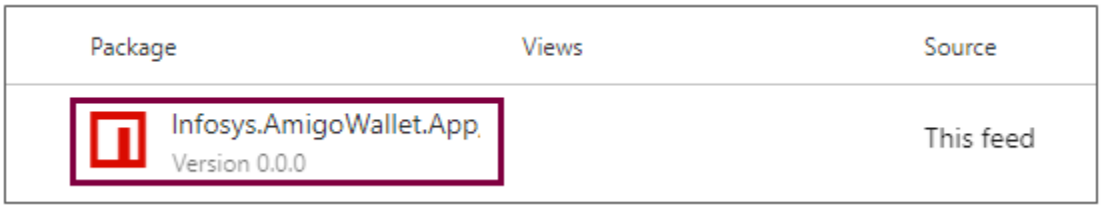
You will see the package appearing here.



Thus the package is created successfully.

Step 9:

In order to consume the package in the angular application, click on the package created.

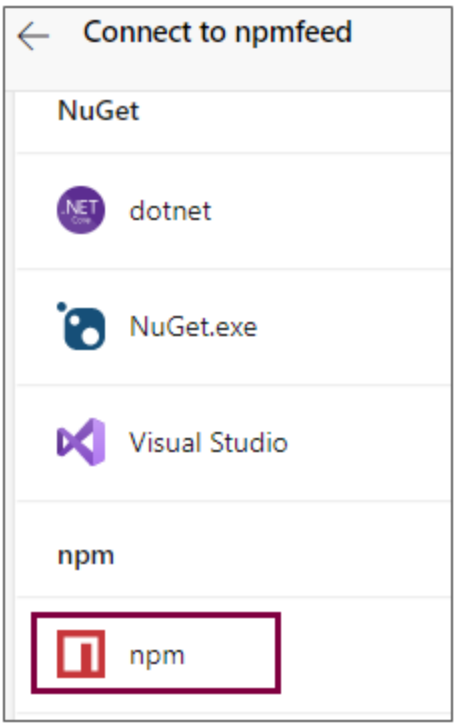


Step 10:

You need to "Connect to feed" to consume the package from the angular application.

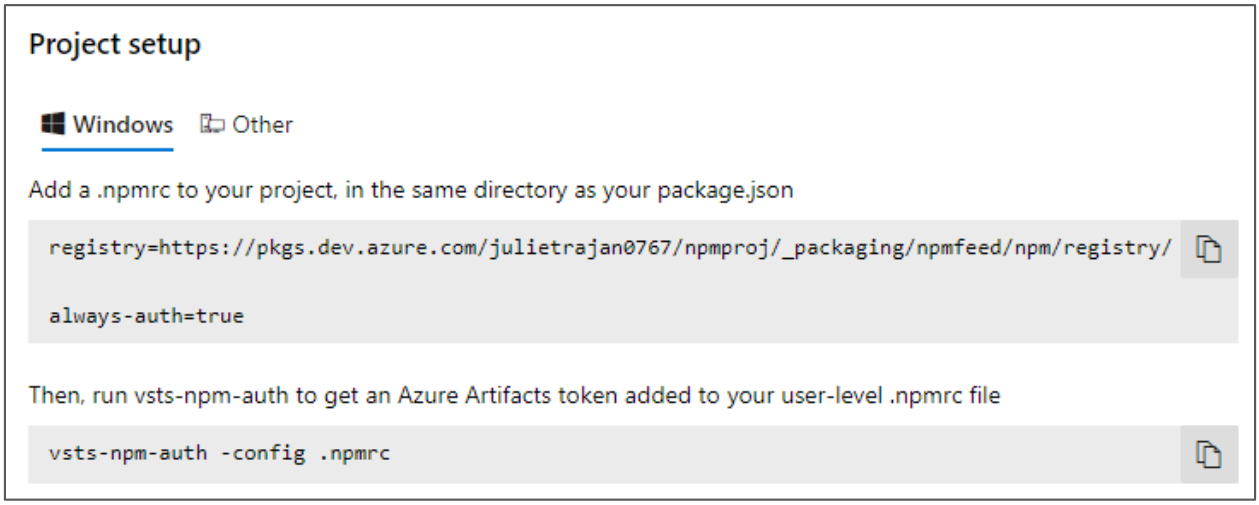
Step 11:

Choose "npm" from the below options



Step 12:

You will see the commands here to consume the npmfeed from your angular application. Execute the commands from your angular cli.


Project setup
Windows
Other
Add a .npmrc to your project, in the same directory as your package.json
registry=https://pkgs.dev.azure.com/julietrajan0767/npmproj/_packaging/npmfeed/npm/registry/
always-auth=true
Then, run vsts-npm-auth to get an Azure Artifacts token added to your user-level .npmrc file
vsts-npm-auth -config .npmrc

On successful execution of the above commands, you will notice the package is added to the angular application.